マイクロプログラム制御計算機の高速化,高信頼化設計に関する研究(Dissertation_全文)

AUTHOR(S):

岩根, 雅彦

マイクロプログラム制御計算機の
高速化、高信頼化設計に関する研究

昭和59年9月

岩根雅彦

マイクロプログラム制御計算機の
高速化、高信頼化設計に関する研究

昭和59年9月19日

岩根雅彦

目　　次

第 1章　まえがき

１．１．はじめに

計算機が一般社会で使用された初期の時代では、計算機の利用形態は、科学技術計算、事務処理等のいわゆるオフラインのバッチ処理が中心であったが、高速化及び信頼性の向上に伴って預金為替業務、座席予約、交通管制等のオンラインリ･アルタイム処理、及び計算機を多くの使用者が時分割で使用するタイムシェアリング処理に拡大された。さらにLSI(Large Soale Integration)技術が進歩し、マイクロコンピュータ（マイコン）が登場し、あらゆる機器に組込まれるようになってきた。

現在では、マイコン、ミニコンピュータ（ミニコン）及び汎用計算機等の多くの計算機が、産業、医療、教育、交通、流通等のあらゆる分野に使用され、今日の産業活動、経済活動、社会活動の多くは計算機の存在の上に組上げられている。

初期の計算機では結線論理方式が多かったが、マイクロプログラム制御方式はIBMシステム/360への採用を契機として一般化し、現在では、マイコン、ミニコン及び

汎用計算機の多くの機種に採用されている。

マイクロプログラム制御方式の特長は、制御回路の設計がプログラミングに置き換わるので、設計、製作及び検査保守等が容易となること、ハードウェアの細部をプログラム制御できるので、設計の自由度が大きく、機能の拡張性、融通性に富むこと、等であり、この特長を生かすことによって、少ない費用で強力な命令セットの実現が可能となり、また制御論理の変更が容易となる。さらに応用面では、エミュレーションの実現、マイクロプログラム化による性能改善、信頼性の向上等がある。

筆者はこのような背景をもつマイクロプログラム制御計算機について、ハードウェア費用を考慮した設計という立場で高速化及び高信頼化の研究に従事してきた。

本章では、マイクロプログラム制御計算機の歴史の概略を眺め、次に、高速化及び高信頼化の研究において、いままでいかなる研究がされ、どのような研究課題が未解決で残っているかを述べ、本研究の意義を明確にするためのバックグラウンドを与える。そして、本研究の目的を述べ、第 2章以下の内容を簡単に紹介する。

１．２．マイクロプログラムに関する研究の歴史

１．２．１．マイクロプログラムの出現

マイクロプログラム制御方式は、1951年にM.V.Wilkesによって計算機の制御部を系統立てて組織的に設計する方法として提案[1]された。その動作原理は次のようなものである[2,3]。

マイクロプログラム制御計算機では、機械命令（以後単に命令と呼ぶ）はレジスタ間の情報の転送やカウンタの計数等の基本的な操作、すなわちマイクロ操作に分解される。同時に発生される制御信号によって実行される幾つかのマイクロ操作をまとめて命令形式にしたものをマイクロ命令とよぶ。計算機のプログラムが命令の系列で構成されるのに対して、命令はマイクロ命令の系列で構成されると考えられ、この系列をマイクロプログラムと呼ぶ。このマイクロプログラムを専用の制御記憶に格納しておき、そのマイクロ命令を順次読み出して、その中のマイクロ操作を実行することにより命令を実行することができる。

この考えに基づいてイギリスのケンブリッジ大学でEDSAC 2[4]なる計算機が開発された。国内では京都大学と東

芝によって1961年に初めてマイクロプログラム制御方式のKTパイロットモデル計算機[5]が試作された。

同年にマイクロプログラムを通常のプログラムと共にコアメモリに格納しておく書換え可能なマイクロプログラム制御方式の計算機AN/UYK-1[6]が発表された。これはTRW-133,330[7]と発展し、国内では、1963年にMELCOM 1530[8]として商品化された。しかしコアメモリの読み出し時間が遅いために命令実行時間が長くなったために発展しなかった。

1964年にIBMシステム/360[9]にマイクロプログラム制御方式が採用され、旧機種のプログラムをそのままの形で効率よく実行するためにエミュレーション技術[10,11]が開発された。IBMシステム/360ではMODEL 30から70までプログラムの互換性を保つように命令体系を一致させ、小型のモデルでも大型と同じ命令体系を経済的に持てるように最上位のモデルを除いてマイクロプログラム制御方式が採用された[12]。この後、RCAスペクトラ/70[13]が発表され、国内では、KTパイロット計算機を製品化したマイクロプログラム制御方式のTOSBAC 3400が商用に供せられて、各社が相次いでマイクロプログラム制御方式の計算機を

発表し、マイクロプログラム制御方式は一般化していった。この頃の制御記憶はキャパシタ、ダイオードマトリックスやコイルカップル等[14]を使用した読み出し専用であった。

1970年代になると、半導体技術が進歩してICメモリが登場し、高速の書き換え可能な制御記憶が実用化され、マイクロプログラムの応用が拡大すると共に、制御記憶の中のマイクロプログラムを書換えて使用するダイナミックマイクロプログラミング技術[15]が開発され、B-1700[16]で実用化された。そしてマイクロプログラムは、処理速度の向上、信頼性の向上及び他の機種との互換性の確保に使用されるようになった。

1971年に 4ビットの汎用マイクロコンピュータINTEL4004[17]が発表されて以来、8,12,16,32ビットへと拡大し、割り込み機能、処理速度の高速化及びアドレッシング機能の向上が行われ、従来のミニコンピュータ以上の能力を持つものまで現れている。マイクロコンピュータの機能向上に伴ってマイクロプログラム制御方式が多く採用されてきている[18]。

１．２．２．高速化の研究

マイクロプログラム制御計算機の処理速度を向上させるためには、高速な論理素子の使用、バッファ記憶の装備、パイプライン処理の採用等のほかに、マイクロプログラム制御方式固有の手段として、マイクロ命令の実行と次のマイクロ命令の読み出しとの並列処理、マシンサイクル時間の可変長化、並列処理可能なマイクロ操作の同時実行、マクロな機能のマイクロプログラム化等の高速化技術がある。

従来の高速化の研究は、並列処理可能なマイクロ操作の同時実行及びマクロな機能のマイクロプログラム化に関するものが中心であり、前者はマイクロプログラムの最適化として、後者は高級言語の処理、オペレーティングシステムのマイクロプログラム化、問題への適応化として研究されている。

マイクロプログラムの最適化[19]のなかで、語数の最適化がマイクロプログラムの実行時間の短縮及び制御記憶に占めるマイクロ命令数の縮小を目的とする研究として1970年代に進展していった。この最適化の初期の研究は、垂直形マイクロプログラムに対してプログラムの最適化手

法を適用する提案[20]が行われ、続いて水平形マイクロプログラムに対してハードウェアリソースの競合とマイクロ操作間のデータ依存性を考慮した最適化手法が高級マイクロプログラム記述言語SIMPLEコンパイラに組み入れられた[21]。この考えに、より現実的な条件を加味した手法が開発されてマイクロプログラムコンパイラに組込まれ実用化[22]されているが、効率の良い最適化アルゴリズムは見つかっていない。

高級言語処理のマイクロプログラム化[23]の実験レベルの顕著な実験例は、1967年にEuler言語をIBMシステム/360モデル30のマイクロプログラムを使用して、中間言語に翻訳して解釈実行するものであった[24]。1972年に、高級言語に応じた中間言語を設定し、この中間言語をマイクロプログラムで解釈実行する商用のB-1700計算機が発表された。以後APL,COBOL,FORTRAN,PASCAL,BASIC,LISP等の高級言語をマイクロプログラムで効率よく処理する実験が行われて発表されている[25-29]。

オペレーティングシステムのマイクロプログラム化の研究は、1972年にInterdata 3を使用して実験的なマルチプログラミングシステム(Venus operating system)で

行われたもの[30]が最初であろう。この後、多くの研究者はマイクロプログラムとオペレーティングシステムとの関係について議論しており、商用の計算機でも割り込み処理、タスクの制御等をマイクロプログラム化しており、IBM システム/38 ではオペレーティングシステムの多くがマイクロプログラム化されている[31]。

問題適応化のおもな研究はアーキテクチャチューニングであり、これは計算機の使用環境に最適になるようにプログラムの一部をマイクロプログラムに置き換えて高速化するものである。アーキテクチャチューニングの研究は、メモリアドレスパターンに注目してチューニングする提案[32]に始まり、命令パターンに注目し命令を合成してチューニングする方法[33]等、その後手法の改良が行われている。また、命令合成の効果をコンパイル時に予測して、効果の大きいものからマイクロプログラム化していく研究[34]が行われている。しかし、自動チューニングの技術はまだ商用の計算機に適用されていない。

1．2．3．高信頼化の研究[35,36]

IBM システム/360が発表される以前のマイクロプログ

ラム制御計算機では、信頼性を向上させるために、信頼度の高い素子の使用、少量の部品による同一機能の実現、データ保全のための 2 out of 5 符号やパリティ符号等の故障検出符号の採用、及び保守のためのソフトウェアによる検査診断プログラムが用意されている程度であった。

IBM システム/360では、故障検出のためにパリティ予測加算器(Parity-predict adders)、相補的論理チェック(Two-rail Logic Checking)、故障診断のためにFLT(Fault Locating Tests)[37]及びマイクロ診断(Microdiagnostics)[38]が実用化された。国内でも FLTはNEAC2200/700[39]、マイクロ診断はHITAC8350[40]、システム/360の下位モデルと同じようなFLT にマイクロプログラムを利用した自動診断方法がTOSBAC5400[41]、等で採用された。

IBM システム/370において、高信頼化はシステム的にRAS(Reliability,Availability,Serviceability)概念[42,43]として統合された。IBM システム/370では、IBM7030 で主記憶に採用された誤り訂正符号より効率のよい誤り訂正符号[44]を採用し、またマイクロプログラムによる命令再実行機能、及びバッファ記憶に自動縮退機構が実用化され

た。[45,46] またマルチCPU 構成では代替CPU 回復機能(Alternate CPU Recovery)がモデル168MP で採用された[47]。さらに、従来の各種パネルの機能と操作卓機能を持ったSVP(Service Processer)がモデル125 において実用化され[49]、国内ではFACOM M190[50]、ACOS 600[51] 等で採用された。

最近ではLSI の発展に伴って、故障検出ではモジュールレベルでのマイクロ命令による照合チェックがACOS800[52] で、故障診断では検査容易性[53] を考慮したLSSD(Level Sensitive Scan Design)[54] がIBM4300 等で実用化されている。また、間欠故障に対して、診断プログラムを繰返し実行させずに、故障検出回路を利用して確率的に局所化する実験[55] も行われている。

マイクロプログラム制御方式がマイコン、ミニコン、汎用計算機の多くに採用され、産業界のあらゆる分野に利用されているので、RAS の向上はますます重要となってきているが、できるだけ少ない費用で同一のRAS 機能を実現することが計算機設計の立場から重要である。

１．３．本研究の概要

１．３．１．本研究の目的

商用の汎用マイクロプログラム制御計算機のハードウェア設計において、性能、信頼性及び費用は重要な設計基準であり、少ない費用でできるかぎり性能を高めかつ信頼性を向上させることが良い設計であると言える。

マイクロプログラム制御計算機の処理速度を向上させるためには、高速な論理素子の使用、パイプライン処理の採用、バッファ記憶の装備等のほかに、マイクロプログラム制御計算機固有の方式上の手段として、マイクロ命令の実行と次のマイクロ命令の読み出しとの並列処理、マシンサイクル時間の無駄時間の削減、並列処理可能なマイクロ操作の同時実行、マクロな機能のマイクロプログラム化等がある。マイクロ命令の実行と読み出しの並列処理、マクロな機能のマイクロプログラム化は多くの商用計算機において採用され効果を上げている。また並列処理可能なマイクロ操作を同時に実行することによって最小のマイクロプログラムを構成する効率の良い最適化技法は見つかっておらず、技術者の経験またはヒューリスッティクな手法で最小に近いマイクロプログラム

を構成して多くの計算機に採用している。この問題は計算量の複雑さに関してNP完全であるので、効率の良いアルゴリズムの開発は困難であることが予想される。マシンサイクル時間の無駄時間の削減は可変長マシンサイクル時間方式とすることによって実現でき多くの商用計算機で採用されているが、そのマイクロ命令実行分布までも考慮した最適化は行われていない。そこで、マイクロ命令実行時間に基づいた最適な可変長マシンサイクル時間を決定するためにアルゴリズムを開発すること、最大公約数的な分布を仮定して最適なマシンサイクル時間を求めて可変長マシンサイクル時間方式の計算機を開発し評価すること、及び使用環境に最適なようにマシンサイクル時間をチューニングする方法を示すことによって、可変長マシンサイクル時間方式の最適設計法を確立することが本研究の目的の一つである。

マイクロプログラム制御計算機の信頼性を向上させるためには、信頼性の高い素子の使用、使用部品数の削減等のほか、マイクロプロラム制御固有の方式上の手段として、マイクロ命令レベルでのオンライン検査プログラムによる故障検出、マイクロプログラムでの再実行によ

る間欠故障の回復、マイクロプログラムによる故障個所の分離及び再構成、及びマイクロ診断プログラムによる故障個所の指摘並びに故障修復等がある。故障回復、再構成及び故障修復の諸手段は、マイクロプログラムとサービスプロセッサとが機能分担することによって実現されている。故障検出、故障回復及び故障修復では、通常の命令実行のために用意されたマイクロ操作のほかに、特別な幾つかのマイクロ操作とハードウェアが必要である。しかし、これらの作業に必要な機能を基本的な共通の小さい機能に分解して、この基本的機能を実行するハードウェアを用意し、基本的機能を組み合わせることによって作業に必要な機能を実現するように設計できるであろう。また CPUの稼動中の遊休時間が頻繁に存在すれば、間欠故障に対してこの遊休時間に検査及び診断プログラムを実行することにより局所化できるであろう。さらにサービスプロセッサは信頼性向上のための重要な機器であるので、ハードウェア量を少なくして機器自身の信頼性を向上させることができよう。このように信頼性確保のためのハードウェアをできる限り少なくして信頼性を向上させることがもう一つの目的である。

１．３．２．各章の概要

本論文は 8章より構成されている。第 2章において、マイクロプログラム制御計算機の高速化及び高信頼化における設計上の問題を論じる。高速化では、平均命令実行時間とマイクロ命令実行時間分布との関係及びマイクロ命令実行時間分布とマシンサイクル時間との関係を調べ、マシンサイクル時間が平均命令実行時間に与える影響について論じる。高信頼化では、マイクロプログラムを利用した故障検出、故障回復及び故障修復技術について述べ、これらに必要なハードウェアの共通化について論じる。

第 3章では、可変長マシンサイクル時間の最適化について論じる。完全可変長マシンサイクル時間方式の最適化には動的計画法による解法が有効であり、また量子化可変長マシンサイクル時間方式の最適化には解の存在範囲の有限性を利用した単純な解法が有効であることを明らかにする。

第 4章では、設計時のある仮定によるマイクロ命令実行分布に基づいたマシンサイクル時間の最適化ついて、

マシンサイクル時間の種類とその時間、性能及び費用の観点から論じ、さらに完全可変長マシンサイクル時間方式の小型計算機を開発した結果について論じる。また使用環境に適するように、マシンサイクル時間の種類を固定し、マシンサイクル時間のみを最適化してチューニングするときの改善率について論じる。

第5章では、RAS向上のためのハードウェア構成について議論する。命令実行のために用意されたマイクロプログラム、RAS向上のための共通化ハードウェアを動作させる診断コマンド及び低速小規模マイクロプロセッサ内蔵のサービスプロセッサ(SVP)の組み合せでもって、ハードウェア量を削減できることを明らかにする。

第6章では、自己検査機構(PATROL)による故障検出技術及びSVPを利用した故障回復技術について議論する。まず、PATROLの基本動作について述べ、実際に使用した結果について検討し、性能に及ぼす影響は無視できる程度で、さらに間欠故障の検出に有効であることを明らかにする。故障回復に必要なSVPのメモリ常駐ソフトウェア量を示し、低速小規模マイクロプロセッサでも実用的であることを明らかにする。

第 7章では、故障修復に必要な保守プロセッサとマイクロ診断プログラムの設計思想と開発結果について論じる。保守プロセッサ及びマイクロ診断プログラムのなかの診断コマンド列とマイクロ命令から構成される CPU駆動手順に低級言語を導入することによって、手順自身がコンパクトになり、手順の開発及び保守が容易になることを明らかにする。さらに、この低級言語の利用と、フロッピーディスクのアクセス回数を少なくすることによって小規模低速マイクロプロセッサでも実用的な SVPが構成できることを明らかにする。

第 8章は本論文の結論であって、本論文に記述されている研究成果と今後に残された問題点を総括して議論する。

１．４．おわりに

本章では、マイクロプログラム制御計算機の歴史を、高速化及び高信頼化の研究という立場から眺めた。

高速化の研究では、マイクロプログラミング自身に関する研究は行われているが、マシンサイクル時間を可変にする研究は殆んど行われていない。また、高信頼化では、RAS を実現するハードウェア量を削減する努力が十分に行われておらず、さらに間欠故障の故障修復技術の研究は余り行われていないことが判った。

最後に、本研究をとりあげるに到った過程と本研究の目的を述べ、第 2章以下各章の概要を述べた。

参考文献

1) M.V.Wilkes:The best way to design an automatic calculating machine,Manchester Univ. Computer naugural Conf.,pp.16-21(951)

2) S.Husson:Microprogramming;Prinoiples and Practices,Prentice Hall(1970)

3) 萩原　宏：マイクロプログラミング、産業図書(1977)

4) W.Renwick:EDSAC 2,Proc.IEE,Vol.103,Pt.B,Suppl.2, pp277-278(1956)

5) H.Hagiwara,K.Amo,S.Matsushita and H.Yamauchi:The KT pilot computer - A microprogrammed computer with a phototransister fixed memory,Proc.IFIP Cong. 62,pp684-689,North-Holland(1963)

6) H.M.Semarne and R.E.Porter:A stored logic computer,Datamation,Vol.7,No.5,pp33-36(May,1961)

7) W.C.McGee:The TRW-133 computer,Datamation,Vol.10 No.2,pp27-29(Feb.1964)

8) 磯部、杉本、南沢、和田編：電子計算機のプログラミング，共立出版(1967)

9) G.M.Amdahl,G.A.Blaauw and F.P.Brooks:Architecture of the IBM System/360,IBM Jour. of R and D, Vol.8,No.2,pp.87-101(Apr.1964)

10) M.A.McCormack,T.T.Sohansman and K.K.Womack: 1401 compatibility feature on the IBM System/360 Model 30,Comm. ACM,Vol.8,No.12,pp773-776 (Dec.196 5)

11) S.G.Tucker:Emulation of large systems,Comm. ACM,Vol.8,No.12,pp.753-761(Dec.1965)

12) S.G.Tucker:Microprogram control for System/360

IBM Syst. Jour.,Vol.6,No.4,pp.222-241(1967)
13) C.R.Campbell and D.A.Neilson:Microprogramming the Spectra 70/35,Datamation(Sept.1966)
14) 萩原：マイクロプログラミングと固定記憶装置、情報処理，Vol.5,No.1,pp31-34(Jan.1964)
15) A.B.Tucker and M.J.Flynn:Dynamic microprogramming - Processor organization and programming, Comm. ACM,Vol.14,No.4,pp240-250(Apr.1971)
16) W.T.Wilner:Design of the Burroughs B1700,AFIP Conf. Proc.,Vol.41,1972 FJCC,pp489-497(1972)
17) 当麻編：マイクロコンピュータ応用ハンドブック，昭晃堂(1983)
18) T.G.Rauscher and P.M.Adams:Microprogramming:A Tutorial and Survey of Recent Developments, IEEE Trans. on Comput.,Vol.C-29,No.1,pp2-20(1980)
19) T.Agerwala:Microprogram Optimizatin:A Survey, IEEE Trans. on Comput.,Vol.C-25,No.10,pp962-973(Oct.1976)
20) R.L.Kleir and C.V.Ramamoorthy:Optomization Strategies for Microprograms,IEEE Trans. on Comput.,Vol.C-20,No.7,pp783-795(Jul.1971)
21) C.V.Ramamoorthy and M.Tsuchiya:A High-level Language for HorizontalMicroprogramming,IEEE Trans. on Comput.,Vol.C-23,No.8,pp791-801(Aug.1974)
22) 松崎：実用化の試みが始まったマイクロプログラムの最適化，日経エレクトロニクス，1978.5.1,pp76-91(1978)
23) 藤野：言語指向型マシンとマイクロプログラム，情

報処理，Vol.14，No.6，pp415-420(1973)
24)H.Weber:A microprogrammed implementation of EULER on IBM System/360 Model 30，Comm. ACM，Vol.10，No.9，pp.549-558(Sept.1967)
25)A.Hassit，J.W.Lageschulte and L.E.Lyon:Implememtation of a high-level language machine，Comm. ACM，Vol.16，pp199-212(1973)
26) 萩原、富田：マイクロプログラム制御計算機QA-1の応用、昭和53年度科研費研究成果報告書(1979)
27)T.R.Bashkov，A.Sasson and A.Kronfeld:System Design of a FORTRAN Machine，IEEE Trans. on EC，Vol.EC-16，No.4，pp485-499(Aug.1967)
28) 島田、山口、坂村：LISPマシンとその評価，信学論 Vol.59-d，No.6，pp406-413(1976)
29)R.Zaks，D.Steingart and J.Moore:A firmware APL time-sharing system，AFIPS Conf.Proc.，Vol.38，1971 SJCC，pp179-191(1971)
30) B.H.Liskov:The Design of Venus Operating System，Comm.ACM，Vol.15，No.3，pp144-149(1972)
31)IBM:IBM System/38 Teohnioal Developments，pp 1-110(1978)
32)A.M.Alla and D.C.Karigaard:Hueristic Synthesis of Microprogrammed Computer Arohteoture，IEEE Trans. on Comput.，Vol.C-23，No.8，pp802-807(1974)
33) 坂村、相磯：計算機アーキテクチャの自動最適化に関する考察，信学論，Vol.60-d，No.11，pp929-936(1977)
34) T.G.Rausoher and A.K.Agrawala:Dynamio Problem Oriented Redefinition of Computer Arohteoture via Mioroprogramming，IEEE，Trans. on Comput.，

Vol.C-27,No.11,pp1006-1014(1978)
35) 猪瀬編：コンピュータシステムの高信頼化，情報処理学会（1977）
36) M.Y.Hsiao,W.C.Carter,J.W.Thomas and W.R.Stringfellow:Reliability,Availability,Serviceability of IBM Computer Systems:A Quarter Century of Progress,IBM Jour. of R and D,Vol.25,No.5,pp453-465(Sept.1981)
37) W.C.Carter,H.C.Montgomery,R.J.Preiss and H.J.Reinheimer:Design of Serviceability Features for the IBM System/360,IBM Jour. of R and D,Vol.8,No.2,pp115-126(Apr.1964)
38) N.Bartow and R.McGuire:System/360 model 85 microdiagnostics,AFIP Conf. Proc.,Vol.36,1970 SJCC,pp191-197(1970)
39) 北村、稲垣：論理装置の故障診断，情報処理，Vol.13,No.9,pp607-616(1972)
40) 酒井：中型計算機HITAC 8350の診断方式，日立評論 Vol.55,No.10,pp13-17(1973)
41) 三浦、高木、斎藤：TOSBAC5400モデル150故障診断システム，東芝レビュー，Vol.28,No.11,pp1252-1257(1973)
42) 北村、富田：RAS技術の現状，情報処理，Vol.12,No.8,pp497-504(1971)
43) 松崎：システム/370におけるRAS機能，情報処理，Vol.13,No.9,pp617-623(1972)
44) M.H.Hsiao:A Class of Optimal Minimum Odd-Weight Column SECDED Codes,IBM Jour. of R and D,Vol.14 No.4,pp395-401(1970)
45) IBM:A guide to the IBM System/370 Model 155

46) IBM:A guide to the IBM System/370 Model 165
47) 松崎：密結合マルチプロセッサの実現手法，日経エレクトロニクス，1976.9.6,pp124-150(1976)
48) 稲葉：マイクロ診断とその実際，情報処理，Vol.14 No.6,pp408-414(1973)
49) IBM:IBM System/370 Model 125 Funotional Characteristios(1972)
50) 山田：コンピュータアーキテクチャ，産業図書(1976)
51) 岩根、飯田、西中：オペレーションプロセッサ，昭52年信学会全国大会，1356
52) 田中、三好、石井、幸野：装置ないしモジュールレベルで二重化を図ったコンピュータ，日経エレクトロニクス，1977.3.21,pp100-121(1977)
53) T.W.Williams and K.P.Parker:Design for Testability - A Survey,Proo. of IEEE,Vol.71,No.1, pp98-112(1983)
54) E.B.Eichelberger and T.W.Williams:A Logic Design Structure for LSI Testability,14th DA conf proc.,pp462-468(1974)
55) D.C.Bossen and M.Y.Hsiao:Model for Transient and Permanent Error-Detection and Fault-Isolation Coverage,IBM Jour. of R and D,Vol.26,NO.1, pp67-77(1982)

第２章　高速化、高信頼化における設計上の問題点

２．１．はじめに

中央処理装置（CPU）の設計時において性能／価格比を良くすることは重要であり、与えられたハードウェア資源でCPUの能力を最大に発揮させる必要がある。そのためにマイクロプログラムを構成している各マイクロ命令の実行時間とその実行頻度から、価格／性能比を考慮して、マイクロプログラムの実行時間を最小にするような数種類のマシンサイクル時間をもつ方式が考えられる。実際、数種類のマシンサイクル時間をもった、TOSBAC3400，TOSBAC5400[11]，IBM4300[1,2]，SYSTEM/38[3] などの商用の汎用計算機が開発されてきた。

しかし、マイクロプログラムの高速化における従来の研究は、使用可能なハードウェア資源とマシンサイクル時間を所与のものとして取扱っており、マシンサイクル時間自身を最適化しようという試みは十分に行なわれていなかった。

CPUの性能を示す尺度[4]として、命令実行時間をギブソ

ンミックスやコマーシャルミックスのような各種命令ミックス[5]で荷重平均して求められる平均命令実行時間、或はその逆数である単位時間あたりの命令実行回数MIPS(Million Instruction Per Second)が用いられている。この尺度は同一方式の計算機に対して比較的簡単にかつ正確に評価できるので、ハードウェア設計時の性能評価に有効である[6]。

これらの命令ミックスから各マイクロ命令の実行頻度を求め、これに基ずいて最適なマシンサイクル時間を決めることができる。また各種応用分野における典型的なプログラムを実際に実行してあるいはシミュレーションにより得られる各マイクロ命令の実行頻度から最適なマシンサイクル時間を決定することにより各々の分野に適したCPUを構成できる。

設計時に仮定したマイクロ命令の実行頻度は、実際にCPUが使用される環境における実行頻度と異なることがあるので、このような使用環境では最適なマシンサイクル時間のCPUであるとは言えない。それゆえ使用環境に適合するようにマシンサイクル時間をチューニングすることが必要である。

従来、設計時のマシンサイクル時間は、技術者の経験により決定されており、効率よく解くアルゴリズムは知られていなかった。この最適な数種類のマシンサイクル時間を決定する問題は組合わせ計画問題[7]であり、マイクロ命令実行時間の種類が多い場合、そのままの形で最適解を求めることは実用的ではない。

CPUの設計において性能と同様に信頼性を高くすることも重要であり、故障が発生したときその影響が最小になるように留意する必要がある。この故障の影響がシステムサービスに与える大きさと、この影響を除去する能力によって信頼度を評価できる。すなわち、ある故障が発生したときに、その故障による悪影響がいろんなところに伝搬する以前の早い時期に故障を検出する能力、この悪影響によるシステムサービスの停止を回避する能力、システムサービス停止を回避できなかった時の修復能力を高めることが重要である。

これらの能力を向上させるための技術は、単にハードウェアそのものの技術だけでなく、マイクロプログラム技術、ソフトウェア技術及び人間工学的な操作性も含めた方式設計上の技術として認識され、RAS(Reliability

,Availability,Serviceability) 技術[8,9]としてIBM370シリーズ以来多くの機種に組込まれてきた。

この RAS技術を組込むためにはコストは当然上昇するのであるが、少ないコストで最大のRAS 技術を組込む努力は必要である。従来の RAS技術は、間欠故障に対する考慮が十分でなく[10]、また RAS技術を支えるハードウェアは個々の技術に適するように設計されていた。

本章では、まずマイクロプログラム制御の高速化について論じる。CPU の性能は平均命令実行時間で表すことができるが、この平均命令実行時間とマイクロ命令実行時間並びに実行頻度との関係について議論し、そしてマイクロ命令実行時間をマシンサイクル時間に割当てるときの問題と実現するための方式について論じる。

次に、高信頼化設計とRAS 技術について論じる。はじめに一般的な故障発生から故障修復までに採用されているRAS 技術について述べる。RAS 技術の中で特にマイクロプログラムを利用した故障検出技術、故障回復技術及び故障修復技術について述べ、これらの技術を支えるハードウェアの共通化について議論する。

２．２．高速化とタイミング設計

２．２．１．マイクロ命令と処理速度

命令ミックス、実測またはシミュレーションなどから得られた各命令（I(i)，i=1，2，--，N）の実行頻度をw(i)，(i=1，2，--，N)、また各命令の実行時間をτ(i)，(i=1，2，--，n)とする。このときの平均命令実行時間uは次式で表わすことができる。

$$u = \sum_{i=1}^{N} w(i)\,\tau(i), \quad \sum_{i=1}^{N} w(i) = 1 \qquad [2.1]$$

このとき、どのような命令ミックスを選ぶか、どのようなプログラムを実行させるかによって、同一のCPUに対して異なる平均命令実行時間が得られる。

マイクロプログラム制御CPUにおいて、各命令はマイクロ命令の組合わせによって実行される。マイクロ命令は、その命令形式によって水平形と垂直形に分類することができる[11]。一般に水平形マイクロ命令では1マイクロ命令でp種類のマイクロ操作を同時に実行できる。垂直形マイクロ命令は通常の命令と同様な命令形式を持ち、基本的には1マイクロ命令でひとつのマイクロ操作が実行できる。すると水平形および垂直形の1マイクロ命令lの実行時間t(l)は各々[2.2]，[2.3]で表わされる。

$$t(l) = \max\ (v(1), v(2), --, v(k), --, v(p)) \qquad [2.2]$$

$$t(l) = v(k) \qquad [2.3]$$

ここで v(k) はマイクロ操作 k の実行時間である。

問題を簡単にするため、v(k)はマイクロ操作 k ごとに固有の一定時間とし、外部との待合わせを伴うようなマイクロ操作は考慮の対象から外すことにする。すると命令 I(i)の実行時間τ(i)は次式で表わすことができる。

$$\tau(i) = \sum_{j \in I(i)} t(j) \qquad [2.4]$$

ここで、 jは命令 I(i)で実行されるマイクロ命令（種類ではない）であり、t(j)はマイクロ命令 j の実行時間である。

平均命令実行時間 u は次式で表わすことができる。

$$u = \sum_{i=1}^{N} w(i)\left(\sum_{j \in I(i)} t(j)\right) = \sum_{k=1}^{n} h(k)t(k) \qquad [2.5]$$

ここで、t(k)はマイクロ命令 k の実行時間、h(k)はマイクロ命令 k の実行回数、 nはマイクロ命令実行時間の種類である。

マイクロ命令実行回数を相対頻度で表わしたとき、平均命令実行時間 u は次式となる。

$$u = h\left(\sum_{i=1}^{n} f(i)t(i)\right) \qquad [2.6]$$

$$h = \sum_{i=1}^{n} h(k), \quad f(i) = h(i)/h$$

２．２．２．可変長マシンサイクル時間方式

CPU のマシンサイクル時間が一定な方式を固定長マシンサイクル時間方式と呼び、幾つかの異なったマシンサイクル時間を持つ方式を可変長マシンサイクル時間方式と呼ぶことにする。

固定長マシンサイクル時間方式では、１マシンサイクル時間をｔとすると、平均命令実行時間ｕは［2.6］より次式で与えられる。

$$u = ht \qquad [2.7]$$

$$t = max(t(1), t(2), --, t(n))$$

ここで、ｔ(ｉ)はマイクロ命令ｉの実行時間である。

可変長マシンサイクル時間方式において、マイクロ命令実行時間の種類と同数のマシンサイクル時間を持てば、性能は最大になる。このとき、固定長マシンサイクル時間方式と可変長マシンサイクル時間方式との性能差ｄｕは次式となる。

$$du = h(t - \sum_{i=1}^{n} f(i)t(i)) \qquad [2.8]$$

すなわち、マイクロ命令実行時間ｔ(ｉ)のばらつきが小さいほど、その性能差は小さくなるが、このばらつきを

小さく設計することが困難な場合がある。

また、マイクロ命令実行時間の種類と同数のマシンサイクル時間を用意することは、タイミング回路が複雑になり経済的でないので、その中から選んだ m種類のマシンサイクル時間をハードウェアによって実現する必要がある。このマシンサイクル時間T(k)の値に何らの条件をつけない方式を完全可変長マシンサイクル時間方式と呼び、マシンサイクル時間T(k)は基本クロックの整数倍であるという条件をつける方式を量子化可変長マシンサイクル時間方式と呼ぶことにする。

可変長マシンサイクル時間方式では、 n種類のマイクロ命令実行時間t(1),t(2),--,t(n) から m種類のマシンサイクル時間T(1),T(2),--,T(m) を選ぶ必要があり、 m種類のマシンサイクル時間で構成されたときの平均命令実行時間u(m)は、次式で表わされる。

$$u(m) = h\left(\sum_{k=1}^{m} F(k)T(k)\right) \qquad [2.9]$$

ここで F(k) はマシンサイクル時間T(k)で実行されるマイクロ命令の実行頻度である。

完全可変長マシンサイクル時間方式において、 m種類のマシンサイクル時間を持つという条件で性能を最大に

することは、平均命令実行時間u(m)を最小にするマシンサイクル時間T(1),T(2),--,T(m)を求めることであり、次式を最小にするt(i,j)を選択することである。

$$u(m) = h \cdot \sum_{j=1}^{m} \{(\max\ t(i,j))(\sum_{i \in M(j)} f(i,j))\} \qquad [2.10]$$

ここで、t(i,j)はマシンサイクル時間T(j)で実行されるマイクロ命令M(j)の実行時間であり、f(i,j)はマイクロ命令実行時間t(i,j)の実行頻度である。

一方、量子化可変長マシンサイクル時間方式において、IBM 4300で適用されているような方式、すなわち各マシンサイクル時間を基本クロックの連続した整数倍に制限すると、m種類のマシンサイクル時間を持つためには次式を満足する必要がある。

$$\left.\begin{array}{l} (p-1)t(c) < t(1) \leq pt(c) \\ (p+m-2)t(c) < t(n) \leq (p+m-1)t(c) \\ T(j) = (p+(j-1))t(c) \qquad (j=1,2,--,m) \end{array}\right\} \quad [2.11]$$

ここで、t(c)は基本クロック周期、t(1)はt(i)の最小値t(n)はt(i)の最大値、pは整数である。

すると平均命令実行時間u(m)は次式となる。

$$u(m) = F(1)pt(c)+F(2)(p+1)t(c)+---$$
$$---+F(m)(p+m-1)t(c) \qquad [2.12]$$

ここで、F(i)は(p-i)t(c)<t(k)≦(p-i+1)t(c)のマイクロ命令実行時間t(k)のマイクロ命令実行頻度である。

すると、平均命令実行時間u(m)が最小となるt(c)を求めることにより性能を最大にすることができる。

２．３．高信頼化設計とRAS 技術

２．３．１．RAS 技術

RAS 技術は信頼性の向上のための方式設計上の技術であり、RAS を従来の平均故障間隔時間（MTBF）及び平均故障修復時間（MTTR）と次式のように対応づけることができる[12]。

$$\left.\begin{array}{l} \text{Reliability} \leftrightarrow \text{MTBF} \\ \text{Availability} \leftrightarrow \text{MTBF}/(\text{MTBF}+\text{MTTR}) \\ \text{Serviceability} \leftrightarrow \text{MTTR} \end{array}\right\} \quad [2.13]$$

計算機システムの稼動状態で故障が検出されてから再び稼動状態に戻るまで、一般に図2.1 に示す故障処理手順がとられる。この手順を実行するために、ハードウェア、マイクロプログラム及びソフトウェアによるRAS 技術が使用されているが、ここではハードウェア及びマイクロプログラムによるRAS 技術について図2.1 に従って述べる。

故障検出は故障処理の第1歩であり、誤り検出符号による検査（例えばデータ経路のパリティ検査並びにレジデュ検査）、制御部のシーケンス検査、時間監視検査、二重化による照合検査及びテストマイクロプログラムによ

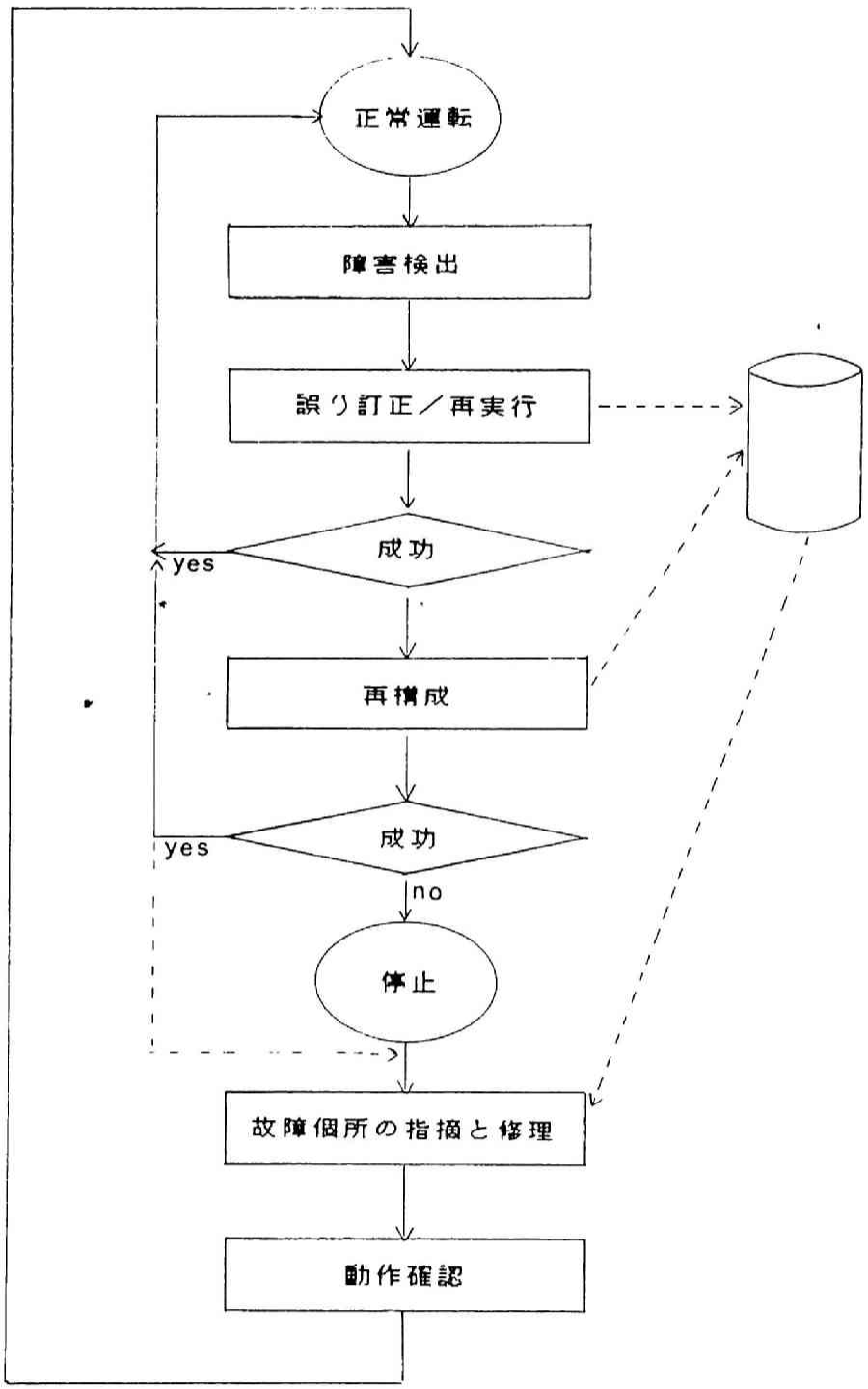


図2.1 故障処理手順

る検査[13]等の方式が採用されている。

誤り訂正及び再実行では、故障による影響が伝搬しないうちに訂正して正常処理を続行しようとすることが試みられており、誤り訂正符号（例えば主記憶及び制御記憶にはハミング符号、入出力関係には巡回符号）を使用して誤り訂正し、マイクロプログラムによって再実行する技術が使用されている。

再構成では、誤り訂正及び再実行で故障回復が不可能であり、かつ故障個所と同一のモジュールまたは装置が複数個あって外部に対して透明であるならば、故障個所をシステムから切り離すことによって稼動状態を保持することが行われる。例えばバッファ記憶並びに高速アドレス変換記憶では故障ブロックをハードウェアによって論理的に切り離したり、マルチCPUシステムでは故障CPUから正常CPUに直接割り込み信号を送るか或はサービスプロセッサSVPに知らせることによって故障CPUを切り離す技術が確立している。

故障修復は故障回復に成功しなかった装置及び切り離された装置を修理してシステムに組込んで稼動状態にすることであるが、これにはSVP並びにマイクロ診断プロ

グラム技術が使用される。

２．３．２．故障検出技術

間欠故障は、固定故障を引き起す過程で起こることが多く、固定故障より少なくとも20倍は多く発生すると言われている[10]。この間欠故障は誤り訂正及び再試行によって救済することができるが、固定故障に変化しシステムダウンを引き起す前に検出修理することが必要である。

そこで、計算機の異常状態をシステムの稼動時に検査するための技法の一つとして、マイクロプログラムレベルの検査プログラムが用意されており、システムの立ち上げ時または周期的に実行されている。これでは、本来カスタマが使用するはずのリソースを検査プログラムが使用することになり、実質的には性能低下となった。すなわち、幾つかのジョブを処理した時のCPU使用時間T'は次式で近似できる。

$$T' = \sum_{i} p(i) + \sum_{j} a(j) + nb \qquad [2.14]$$

ここで、p(i)はCPUが何らかの処理をしている時間、a(j)は遊休状態にいる時間、bは検査プログラムの一回の実行時間、nは検査プログラムの実行回数である。

ところで、CPUが遊休状態の時に検査プログラムを実行すると、CPUの使用時間T'は次式となる。

$$T' = \sum_{i} p(i) + \sum_{j} a(j) \qquad [2.15]$$

一回の検査プログラムの実行時間bを、遊休時間a(j)に等しくなるように設計することによって検査プログラムによる影響を最小にでき、T' < T である。

さらに、マイクロプログラムレベルの検査プログラムでは、検査プログラムを格納するための制御記憶が必要である。このときに必要な制御記憶の語数をw,一語のビット数をdとすると制御記憶の増加分Cは次式となる。

$$C = wd \qquad [2.16]$$

そこで、機能マイクロプログラムを検査プログラムに積極的に利用することを考え、パスファインダメモリ[14]のように間接的に機能マイクロプログラムの命令を指定し実行するように設計すると間接指定記憶の容量C'は次式となる。

$$C' = w'l \qquad [2.17]$$

ここで、w'は間接指定記憶の語数、lは一語のビット数である。

間接指定記憶の1番地で制御記憶の1番地を実行する

ように設計するとw'はwと等しく、複数の番地を実行するように設計するとw'<wが成り立つ。また lは dの数分の一であり、明らかに C'<C が成り立つ。

２．３．３．故障修復技術

故障診断技術では、FLT (Fault Locating Tests)[15]、LSSD(Level Sensitive Scan Design)[17]及びマイクロ診断(Microdiagnostics)[16]が確立している。

FLT 及びLSSDではスキャンイン／スキャンアウト(scan-in/scan-out)のための特別なハードウェアが必要となり、また１クロックの動作で１ビットのデータの読み書きしかできなかった。

マイクロ診断では、スタートスモール(Start Small)による手順で診断を進めていくので、この手順に適した設計を行う必要があるが、機能マイクロプログラムに用意されたマイクロ操作を組み合わせて診断用マイクロプログラムを設計するので、FLT 及びLSSDほどの特別なハードウェアを必要としなかった。しかし、マイクロ診断では制御用順序回路のような複雑な部分に対して直接データを読み出せる構造が必要であり、多数の読み出し用

信号線が必要であった。この制御用順序回路の状態を読み出すために、バス構成をとることによって信号線を削減できる。

マイクロ診断，FLT及びLSSDで使用される検査データは、一般に単一固定故障を仮定して作成されるので、間欠故障や多重故障などの仮定外の故障に対して正しく故障個所を指摘できないことがある。このような時、少し手間どっても保守作業者が故障個所を局所化できる補助機能が必要である。

システム立ち上げ、故障検出、再試行、再構成、故障診断は保守用ハードウェアを使用して行われるが、それぞれ必要な機能が異なるので、十分共通化されておらずハードウェア量は多くなっていた。そこで、故障修復のために必要な基本的な機能に分解すると、その機能は、検査データを適当なレジスタに設定できること、適切な条件でハードウェアを駆動できること、及び検査結果を読み出せることである。

これらの機能をコマンド形式とすることによって、また間接指定記憶を多目的に使用することによって保守用ハードウェアを削減できることが予想される。

２．３．４．サービスプロセッサ

サービスプロセッサSVP（SerVice Processor）[18]は、CPUとは独立したコンピュータシステムで、何んらかの形でCPUと接続されている。一般に、SVPは、プロセッサ、記憶装置、フロッピーディスク、ディスプレー装置、CPUインターフェイス等で構成されている。SVPのプロセッサには、LSI-11，0.2MIPSのマイクロプロセッサのような高性能のマイクロプロセッサまたはミニコンピュータが採用され、記憶装置は32KB～128KBの容量をもっていた[19,20]。

SVPは、システム操作卓機能のほか、システム異常状態監視機能、再試行機能、構成制御機能、保守診断機能、及びシステム立ち上げ機能等を持ち、システム運用及び保全を行うための重要な役割を果たす処理装置であるので、できるだけ簡潔な信頼性の高い構成にする必要がある。そのために部品数が少ない構成にすることが重要であり、このことはコストダウンにもなる。

また、これらの機能の能力は、SVPの内蔵プロセッサのソフトウェアによって決まるが、従来のSVPは内蔵プ

ロセッサの能力を十分に生かしきっておらず、次の欠点をもっていた。

ハードウェア或はソフトウェアの開発または改良中のデバッグ時、並びに複雑な故障の修復時では、CPUを停止させる度にCPU内レジスタの内容を調べることによって動作確認を行うことがある。フロッピーディスクのみを補助記憶装置として持ったSVPでは、レジスタの指定は、理解しやすいニーモニックコードではなく、番号で行われるので操作性が良いとは言えないが、この欠点はソフトウェアを工夫することによって解決できるであろう。また、CPUとSVPは1対1に対応しておりマルチCPUシステムのとき、CPUと同数のSVPが必要となるが、SVPに複数のCPUインターフェイスを設け、システム操作卓機能と保守診断機能を時分割で処理することによって、1つのSVPでマルチCPUシステムをサポートすることが可能であろう。

部品数を少なくするために、低速小規模マイクロコンピュータを内蔵したSVPでも、そのソフトウェアを工夫することによって操作性を良くすることができ、結果としてMTTRを短縮できることによってアベイラビリティを

向上できると推測される。

２．４．おわりに

本章では、マイクロプログラム制御計算機の高速化及び高信頼化設計における問題点について述べた。

高速化設計において、マイクロ命令実行時間のばらつきが大きい程、マイクロ命令実行時間に適した数種類のマシンサイクル時間を持った可変長マシンサイクル時間方式は、マイクロ命令実行時間の最大値を１マシンサイクル時間とした固定マシンサイクル時間方式よりも、有利である。可変長マシンサイクル時間方式には、完全可変長マシンサイクル時間方式と量子化可変長マシンサイクル時間方式があるが、両方式ともこのマシンサイクル時間を決定するときに、マイクロ命令の実行頻度を考慮すれば、さらに性能向上することが期待できる。

高信頼化設計において、マイクロプログラムを利用した故障検出技術、故障回復技術及び故障修復技術について議論した。故障検出技術では、機能マイクロプログラムの一部を間接的に指定して検査プログラムを構成し、それを CPUの遊休時に実行させることによって故障検出のためのハードウェア量及びリソース使用量を少なくできる。故障回復及び故障修復では、SVP が重要な役目を

もつが、SVPの部品数を少ない構成にすることによって信頼性が向上できる。さらに、故障検出、故障回復及び故障修復に必要な機能を基本的な最小機能に分解し、この最小機能をコマンド形式にまとめて共通に使用することによってハードウェア量を少なくできる。

参考文献

1)IBM:A Guide to the IBM4331 Processor(1979)

2)IBM:A Guide to the IBM4341 Processor(1979)

3)F.G.Soltis and R.L.Hoffman:Design Consideration for the IBM System/38,IEEE Compcon'79 proc., pp132-137(1979)

4)萩原：計算機システムの評価について，情報処理，Vol.13,No.11,pp740-745(1972)

5)高橋：コンピュータ評価のための各種ミックス，情報処理，Vol.13,No.11,pp777-781(1972)

6)関野，徳永：システム評価技術，信学誌，Vol.62,No.11,pp1269-1274(1979)

7)西川，三宮，茨木：最適化，岩波書店(1982)

8)IBM:A guide to the IBM System/370 Model155(1970)

9)IBM:A guide to the IBM System/370 Model165(1970)

10) 当麻：高信頼化技術、信学誌，Vol.62,No.11,pp1260-1268(1979)

11) 萩原：マイクロプログラミング，産業図書(1977)

12) 元岡編：計算機システム技術，オーム社(1973)

13) 東芝：TOSBACシリーズ7/40中央処理装置　命令語編

14)A.Grasselli:The Design of Program-modifiable micro-programmed Control Units,IRE Trans. on EC Vol,EC-11,pp336-339(1962)

15)W.C.Carter,H.C.Montgomery,R.J.Preiss and H.J.Reinheimer:Design of Serviceability Features for the IBM System/360,IBM Jour. of R and D, Vol.8,No.2,pp115-126(Apr.1964)

16)N.Bartow and R.McGuire:System/360 model 85 microdiagnostics,AFIP Conf. Proc.,Vol.36,1970

SJCC,pp191-197(1970)
17)E.B.Eichelberger and T.W.Williams:A Logic Design Structure for LSI Testability,14th DA conf proc.,pp462-468(1974)
18)IBM:IBM System/370 Model 125 Functional Characteristics(1972)
19) 中島，西田：下位機種コンピュータにおけるサービスプロセッサの実現手法，日経エレクトロス，1978.5.29,102-119(1978)
20)DEC:VAX11/780 HARDWARE HANDBOOK(1978)

第３章　処理速度向上のための最適化技法

３．１．はじめに

前章で述べたように、可変長マシンサイクル時間方式では、マシンサイクルの種類とその時間が処理速度に大きく影響する。

完全可変長マシンサイクル時間方式におけるマシンサイクル時間の最適化問題を一般化すると、次のような組み合わせ計画問題となり、このままの形で解くことは計算量が大きすぎて実用上不可能である[1]。

長さa(1),a(2),--,a(n)の鋼板において、それぞれb(1),b(2),--,b(n)枚の需用が見込まれるとき、ストックしている長さ d(1),d(2),--,d(m) (n>m)の鋼板一枚から必要な鋼板一枚だけを切り取るときにむだが最小になるように鋼板の長さ d(1),d(2),--,d(m)を決める標準化問題を考える。完全可変長マシンサイクル時間方式におけるマシンサイクル時間の最適化は、この標準化問題に帰着できる。組み合わせ計画問題の解法には、動的計画法、分枝限定法、整数計画法、近似解法等があるが、ナップザック問題、最短経路問題及び巡回セールスマン問

題等は動的計画法によって解くことができる[2]。

量子化可変長マシンサイクル時間方式におけるマシンサイクル時間の最適化問題は、マシンサイクル時間は基本クロックの整数倍の値しかとらないので、単純なアルゴリズムで解くことができる。

本章では、まず完全可変長マシンサイクル時間方式の最適化問題を定式化する。この問題の最適解の性質を調べることによって、計算量の少ない問題に変換する。さらに、この変換問題の最適解の性質を詳しく調べることによって、逐次決定法並びに動的計画法でこの問題が解けることを論じる。そして、完全可変長マシンサイクル時間方式の動的計画法による最適化アルゴリズムと量子化可変長マシンサイクル時間方式の単純アルゴリズムについて述べる。

３．２．完全可変長マシンサイクル時間方式問題

３．２．１．定式化と変換問題。

2.2.2 節の議論より、マイクロ命令a(i)の実行時間をs(a(i))，その頻度をq(a(i))，nをマイクロ命令の種類、mをマシンサイクル時間の種類、そして各マイクロ命令は異なる実行時間を持つものとすると、次式のように定式化できる。

n個の要素 a(1)，a(2)，----，a(n)からなる集合Aとその要素の各々に自然数 s(a(i))，q(a(i))及び、m (m<n) が与えられているものとする。

$$A=\bigcup_{i=1}^{m} A(i),\ A(i)\subseteq A,\ A(i)\neq\phi\ (i=1,2,--,m);$$

$$a(i)\neq a(k),\ s(a(i))\neq s(a(k))\ (i\neq k);$$

$$s(A(j))=\max_{a(i)\in A(j)} s(a(i)),\ q(A(j))=\sum_{a(i)\in A(j)} q(a(i)) \qquad [3.1]$$

とするとき、

目的関数 u=s(A(1))q(A(1))+s(A(2))q(A(2))+--

-+s(A(m))q(B(m))を最小にする問題に一般化できる。

ここで、uを最小とするAの部分集合 A(1)，A(2)，--，A(m) を最適解と呼び、(A(1)，A(2)，----，A(m))で表わすと、この最適解に対して次の定理が成立する。

（定理１） (A(1)，A(2)，-------，A(m))は集合Aの分割で

ある。

（証明）　(A(1),A(2),------,A(m))がAの分割でないと仮定する。すると、　A(i)∩A(j)≠φ　となるような部分集合　A(i),A(j)が存在する。a(k)∈ A(i)∩A(j)　なる要素a(k)が存在し、a(k)を　A(i)　から除いた部分集合をA'(i)とする。そしてA(1),A(2),----,A(m)　に対する目的関数値を　u、A(1),---,A'(i),-----,A(m)　に対する目的関数値をu'とする。すると、u>s(A(1))q(A(1))+s(A(2))q(A(2))+--+s(A(m))q(A(m))-s(A(i))q(a(k))=u'　となり、uが最小値であることに矛盾する。一方、$\bigcup_{i=1}^{m} A(i)=A$ は明らかである。ゆえに　(A(1),A(2),---,A(m))　は　Aの分割である。（証明終）

（定理２）　(A(1),A(2),-----,A(m))　の任意の２つの部分集合を　A(k),A(l)とすると、∀a(i)∈A(k)，∀a(j)∈ A(l) [s(a(i))<s(a(j))]　、または　∀a(i)∈A(k)，∀a(j)∈A(l) [s(a(i))>s(a(j))]　が成立する。

（証明）　(1)s(A(k))<s(A(l))　の場合。s(a(j))<s(A(k))なる　a(j)∈A(l)が存在すると仮定する。この　a(j)　を　A(l)からA(k)に移した部分集合をA'(k),A'(l)とする。A(1),---,A(m)　に対する目的関数値を　u、A(1),---,A(k-1

),A'(k),A(k+1),---,A'(l),A(l+1),----,A(m) に対する目的関数値をu'とする。すると、u - u' =s(A(k))q(A(k))+s(A(l))q(A(l))-s(A(k))(q(A(k))+q(a(j)))+s(A(l))(q(A(l))-q(a(j)))=(s(A(l))-s(A(k)))q(a(j))。ところで s(A(k))<s(A(l))であるので、 u)u'となり uが最小値であることに矛盾する。ゆえに任意の a(j)∈A(l) に対して s(a(j))>s(A(k))である。(2) s(A(k))>s(A(l)) の場合も(1)と同様に任意の a(i)∈A(k) に対してs(a(i))>s(A(l)である。(3) s(A(k))=s(A(l)) の場合は、s(a(i)) ≠ s(a(j)) (i≠j)であるので、存在しない。ゆえに(1)(2)(3)より定理2は成立する。(証明終)

元の問題は要素a(i)をs(a(i)) の小さい順に並べた集合Aを目的関数 uが最小となるようにm-1 個の仕切を入れて分割する問題に変換できる。この仕切を入れて分割する問題を変換問題と呼ぶことにする。

ここで目的関数u を評価する回数を調べてみる。元の問題において、目的関数u の評価回数は、 n種類の区別できる要素を m個の区別できない箱に空箱ができないように入れる入れ方と同じであり、その数は次の漸化式で与えられるスターリング数 $S(n,m)^{3}$ である。

$$\left.\begin{aligned} &S(j,1)=S(j,j)=1 \\ &S(j+1,k)=S(j,k-1)+kS(j,k)\quad (1<k<j) \\ &\qquad\qquad (j=2,3,--,n;k=1,2,--m) \end{aligned}\right\} \quad [3,2]$$

一方変換問題において、評価回数は、s(a(i)) の小さい順に並べた集合A に m-1個の仕切を入れる入れ方と同じで C(n-1,m-1) である。そこで、以後の議論では s(a(i))<s(a(j)) (i<j) と仮定した変換問題を取扱う。以後の議論において、(A(1),A(2),--,A(m)) を変換問題の最適解とする。

３．２．２．変換問題における最適解の性質

（定理３）(A(1),A(2),----,A(m)) の任意の 2つの部分集合A(i),A(j) において、s(A(i))<s(A(j)) (i<j) とする。 (A(1),A(2),----,A(m))のなかの k個の部分集合 A(i_1),A(i_2),--,A(i_k) を取る。この部分集合の和集合 B=A(i_1)UA(i_2)U--UA(i_k) を目的関数 u が最小となるように k分割するとき、このときの最適解を、 (B(1),B(2),--,B(k))とすると、B(1)=A(i_1),B(2)=A(i_2),--,B(k)=A(i_k) である。

（証明）集合 B を k分割したとき、A(i_1),A(i_2),--,A(i_k

) とは別の最適解 $(A'(i_1), A'(i_2), --, A'(i_k))$ が存在すると仮定する。 $A(1), A(2), ----, A(m)$ に対する目的関数値を $u(1)$, $A(i_1), A(i_2), --, A(i_k)$ に対する目的関数値を $u(2)$, $A'(i_1), A'(i_2), --, A'(i_k)$ に対する目的関数値を $u(3)$, $A(1), A(2), --, A(i_1-1), A'(i_1), A'(i_2), --, A'(i_k), A(i_k+1), ---, A(m)$ に対する目的関数値を $u(4)$ とする。

$u(1) = s(A(1))q(A(1)) + --+s(A(i_1-1))q(A(i_1-1)) + u(2) + s(A(i_k+1))q(A(i_k+1)) + --+s(A(m))q(A(m))$

$u(4) = s(A(1))q(A(1)) + --+s(A(i_1-1))q(A(i_1-1)) + u(3) + s(A(i_k+1))q(A(i_k+1)) + --+s(A(m))q(A(m))$

仮定より、$u(3)$ は最小値であるので、$u(3) \leq u(2)$ である。ゆえに $u(4) \leq u(1)$ となる。ところが $u(1)$ は最小値であるにもかかわらず、$u(4)$ が存在することになり矛盾する。よって $A(i_1), A(i_2), --, A(i_k)$ が最適解である。

（証明終）

以後の議論において、定理3の中で明示したように $(A(1), A(2), --, A(m))$ の任意の 2つの部分集合 $A(i), A(j)$ をとったとき、$s(A(i)) < s(A(j))$ $(i<j)$ であるとする。

（定理4） $(A(1), A(2), ----, A(m))$ のなかの $k-1$ 個の部分集合 $A(i), A(i+1), ----, A(i+k-2)$ と $A'(i+k-1) \leq A(i+k-$

1)なる部分集合 A′(i+k-1)をとる。これらの部分集合の和集合 B=A(i)UA(i+1)U--UA(i+k-2)UA′(i+k-1)を目的関数uが最小となるようにk分割するとき、s(B(k-1))(s(A(i+k-2))を満たす最適解 B(1),B(2),---,B(k) が存在する。 ここで、

$$s(B(k-1)) = \underset{a(i)\in B(k-1)}{MAX}\ s(a(i)),$$

$$s(A(i+k-2)) = \underset{a(i)\in A(i+k-2)}{MAX}\ s(a(i)),\ \text{とする。}$$

（証明） Bをk分割したときのどんな最適解も s(B(k-1))≦s(A(i+k-2))を満たさないと仮定する。部分集合A(i),--,A(i+k-2),A(i+k-1) に対する目的関数値を u(1)，B(1),---,B(k-1),B(k) に対する目的関数値を u(2)，A(i),---,A(i+k-2),A′(i+k-1)に対する目的関数値を u(3)，B(1),---,B(k-1),B′(k) (B′(k)=B(k)U(A(i+k-1)-A′(i+k-1))) に対する目的関数値を u(4) とする。そして、

$$q(B(j)) = \sum_{a(i)\in B(j)} q(a(i))$$

$$q(A(j)) = \sum_{a(i)\in A(j)} q(a(i))$$

とすると、

$$u(1)=u(3)+(s(A(i+k-1))-s(A'(i+k-1)))q(A'(i+k-1))+s(A(i+k-1))(q(A(i+k-1))-q(A'(i+k-1)))$$

$$u(4)=u(2)+(s(B'(k))-s(B(k)))q(B(k))+s(B'(k))(q(B'($$

k))-q(B(k)))

u(1)-u(4)=u(3)-u(2)+s(A(i+k-1))q(A(i+k-1))-s(B(k))q(A'(i+k-1))+s(B(k))q(B(k))-s(B'(k))q(B'(k))

ところで、A(i+k-1)-B'(k)=A'(i+k-1)-B(k)，s(B'(k))=s(A(i+k-1))，s(A'(i+k-1))=s(B(k))であるので、上式は次のように変形できる。

u(1)-u(4)=u(3)-u(2)+(s(A(i+k-1)-s(B(k))(q(A'(i+k-1))-q(B(k)))

ところが、仮定よりs(B(k-1)))s(A(i+k-2)であるのでB(k)⊆A'(i+k-1)⊆A(i+k-1) となり、s(A(i+k-1))-s(B(k))≧0，q(A'(i+k-1))-q(B(k))≧0，また、u(3)-u(2)≧0であるので、u(1))u(4) となり、u(1)が最小値であることに矛盾する。ゆえに、s(B(k-1))<s(A(i+k-2)) である。（証明終）

(A(1)，A(2)，--，A(m)) の任意の部分集合A(j)のすべての要素a(j_k) をs(a(j_k))の小さい順に並べると、a(j_1)，a(j_2)，--，a(j_ℓ)であるとする。A(j)を 2つに分割したとき、各々A'(j)，A''(j)とする。 ∀a(i_j)∈A'(j)，∀a(i_k)∈A''(j) [s(a(i_j)))s(a(i_k))]であるとき、A'(j) をA(j)の上位の部分集合、A''(j)をA(j)の下位の部分集合と呼

ぶことにする。

（定理５）（A(1),A(2),---,A(m)）のなかのk-1個の部分集合A(i-k+2),A(i-k+3),---,A(i)とA'(i-k+1)⊆A(i-k+1)なるA(i-k+1)の上位の部分集合A'(i-k+1)をとる。これらの部分集合の和集合B=A'(i-k+1)UA(i-k+2)U---UA(i)を目的関数uが最小となるようにk分割するとき、s(A(i-k+1))≦s(B(i-k+1))を満たす最適解B(i-k+1),B(i-k+2),--,B(i)が存在する。（証明略）

（定理６）集合Aを目的関数uが最小となるようにm分割したときの最適解（A(1),A(2),---,A(m)）において、

Aを同様にm+1分割したとき、s(A'(1))≦s(A(1))≦s(A'(2))≦--≦s(A'(m-1))≦s(A(m-1))≦s(A'(m))≦s(A(m))=s(A'(m+1))を満たす最適解（A'(1),----,A'(m),A'(m+1)）が存在する。

（証明）i個の部分集合A(1),A(2),---,A(i)の和集合B(i)=A(1)U---UA(i-1)UA(i)とi個の部分集合C(i)=A'(1)U---UA'(i-1)UA'(i)を考える。ただしi≦mとする。集合B(m),C(m)を各々目的関数uが最小になるようにm分割すると、s(A'(m))≦s(A(m))であるので定理４よりs(A'(m-1))≦s(A(m-1))である。次に集合B(m-1),C(m-1)を同

様に m-1分割すると、s(A'(m-1))≦s(A(m-1))であるので定理4よりs(A'(m-2))≦s(A(m-2))である。以下同様に、s(A'(m-3))≦s(A(m-3)),---,s(A'(2))≦s(A(2)),s(A'(1))≦s(A(1))である。

次にi個の部分集合A(m-i+1),A(m-i+2),---,A(m-1),A(m)の和集合D(i)=A(m-i+1)UA(m-i+2)U---UA(m)とi個の部分集合A'(m-i+2),---,A'(m),,A'(m+1)の和集合E(i)=A'(m-i+2)U---UA'(m)UA'(m+1)を考える。ただし 1≦i≦mとする。集合D(m),E(m)を各々目的関数uが最小になるようにm分割すると、定理5より、s(A(1))≦s(A'(2))である。次に集合D(m-1),E(m-1)をm-1分割するとs(A(1))≦s(A'(2))であるので定理5よりs(A(2))≦s(A'(3))である。以下同様に、s(A(3))≦s(A'(4)),---,s(A(m-1))≦s(A'(m))である。ゆえに s(A'(1))≦s(A(1))≦s(A'(2))≦--≦s(A'(m-1))≦s(A(m-1))≦s(A'(m))≦s(A(m))=s(A'(m+1))である。(証明終)

３．３．最適化技法

３．３．１．逐次決定法

集合Ａを目的関数ｕが最小となるように ｍ分割したときの最適解 (Ａ(１),Ａ(２),--,Ａ(ｍ))、同様に ｍ+１分割したときの最適解 (Ａ'(１),Ａ'(２),--,Ａ'(ｍ),Ａ'(ｍ+１)) において、集合Ａの部分集合 Ｂ(ｋ)=Ａ(１)∪Ａ(２)∪--∪Ａ(ｋ-１)∪Ａ(ｋ) 及び Ｂ'(ｋ)=Ａ'(１)∪Ａ'(２)∪--∪Ａ'(ｋ-１)∪Ａ'(ｋ)、(ｋ=１,２,--,ｍ)をとる。部分集合Ｂ(ｋ)及びＢ'(ｋ) を目的関数が最小になるように ｋ分割したとき、定理３より最適解は、(Ａ(１),Ａ(２),--,Ａ(ｋ)) 及び (Ａ'(１),Ａ'(２),--,Ａ'(ｋ))である。すると (Ａ'(１),Ａ'(２),--,Ａ'(ｋ))の目的関数ｕ'(ｋ) について次式が成立する。

$$u'(k)=u'(k-1)+s(A'(k))q(A'(k)) \qquad [3.3]$$

また定理６より、$s(A(k-1))\leq s(A'(k))\leq s(A(k))$が成立する。そこで、ｋ=１,２,--,ｍの順序で $s(A(k-1))\leq s(a(i))\leq s(A(k))$を満たす要素ａ(ｉ)毎にｕ'(ｋ) が最小となるようにｕ'(ｋ-１) を計算していくと最適解 (Ａ'(１),Ａ'(２),--,Ａ'(ｍ+１))が求まる。この方法は [３.３]式を |Ａ(ｋ-１)|･|Ａ(ｋ)| 回評価して |Ａ(ｋ)|個の候補を選ぶ繰返しとなる。目的関数の評価回数は、|Ａ(ｋ)|<ｎ,|Ａ(ｋ-１)|<ｎ であるので、ｏ(

m*n**2)[4]である。

３．３．２．動的計画法（Dynamic Programming）[5]

任意の要素数 n及び分割数 mについて定理３が成立するので、分割数を各段にとった図3.1に示す多段ネットワーク[6]を考える。図3.1において、各頂点v(j,k)は j個の要素からなる集合を目的関数uが最小となるようにk(k<j)分割した状態、各辺はある頂点から他の頂点に移るときの目的関数値の増分とする。そして各頂点における目的関数値を u(j,k)、各辺に与えられた増分を d(l,j)とおくことによって、次の漸化式と初期条件が導かれる。ここで各段のどの辺を選ぶかを「決定」、その結果得られる始点からの路を「方策」、各頂点を「状態」とみなすと、順方向の最適性の原理が成立していることは明らかである。

$$
\left.
\begin{aligned}
& u(j,1) = d(1,j) \\
& u(j,k) = \min_{K \le l \le j} [u(l-1,k-1)+d(l,j)] \\
& d(l,j) = s(a(j))(q(a(l))+q(a(l+1))+\cdots \\
& \qquad \cdots+q(a(j))) \\
& (j=1,2,\cdots,n;\ k=2,3,\cdots,m;\ j>k)
\end{aligned}
\right\} \quad [3.4]
$$

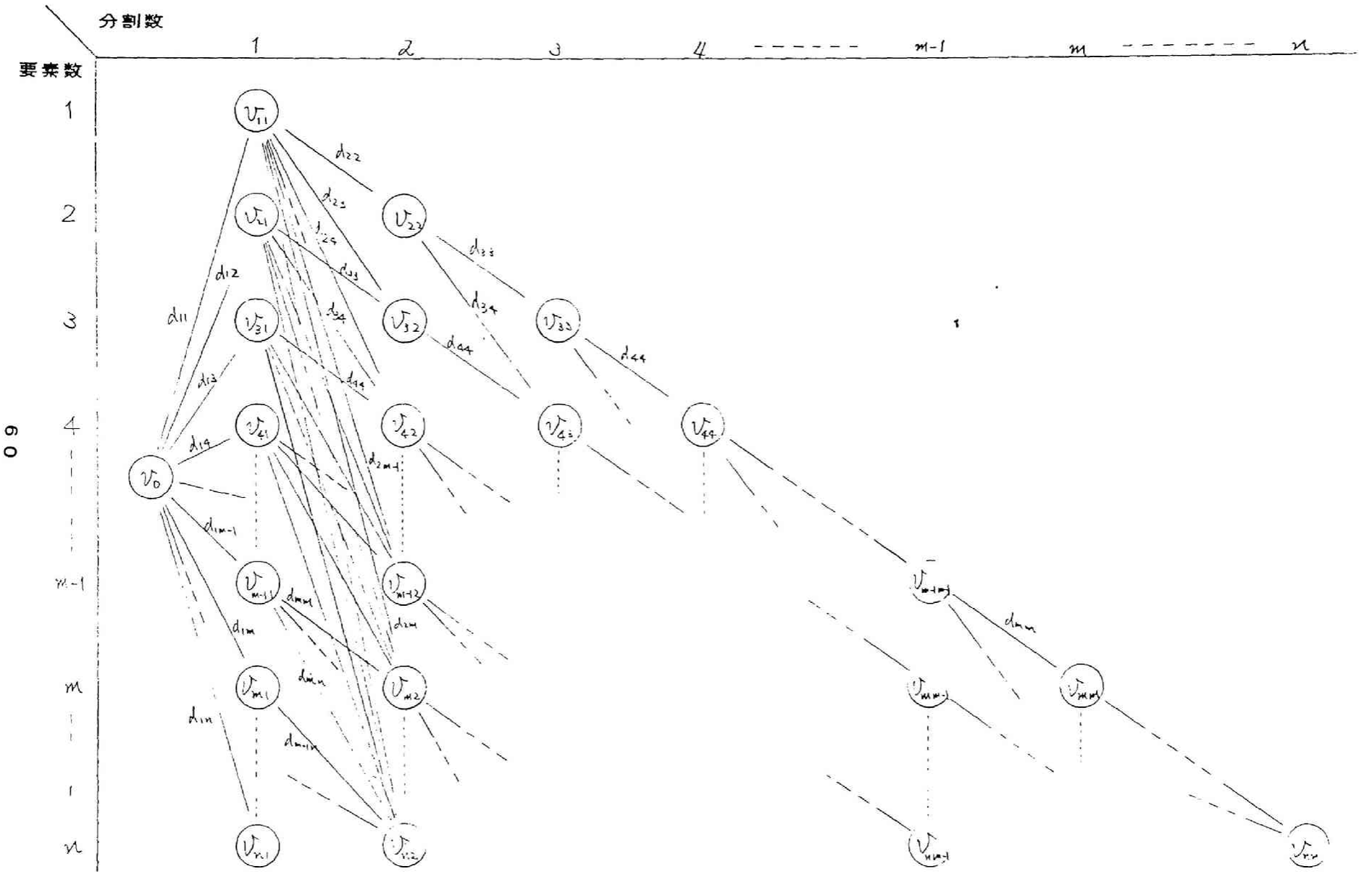


図3.1 多段ネットワーク

ゆえに、多段ネットワークの最短路問題を順方向最適性の原理を具現化した動的計画法（DP）によるものと同じ手法により、この問題を解くことができる。

次に目的関数u(j,k)の右辺の評価回数は、第1段ではn回、第k段では（1+2+--+(n-k+1)）回（k=2,3,--,m-1）、第m段ではn-m+1回の評価が必要であるので、全体の評価回数Tは次式となる。

$$T = n + \sum_{k=2}^{m-1} (1+2+--+(n-k+1)) + (n-m+1)$$

$$= (m-2)*n**2/2 - m*(m-4)*n/2 + m*(m**2-6*m+5)/6 \qquad [3.5]$$

動的計画法の適用によって、目的関数の評価回数はC(n-1,m-1)からTに削減できた。ここでは最適解の範囲を限定することによって評価回数をもっと少なくする。定理6により、集合Aを目的関数uが最小となるようにm分割したときの最適解からm+1分割したときの最適解の範囲が限定できるので漸化式の評価回数を減少することができる。すなわち評価回数Tは次式となる。

$$T < (m-2)*n**2/2 - m*(m-4)*n/2 + m*(m**2-6*m+5)/6 \qquad [3.6]$$

３．４．アルゴリズムと数値例

３．４．１．完全可変長方式の動的計画法による最適化

層別化により最適解の範囲を限定した動的計画法によるアルゴリズムを次に示す[7]。

```
 10  begin
 20    Ａの要素を s(a(i))の小さい順に並べかえ
 30    それをa(1),a(2),--,a(n) とする。
 40    for j=1 until n do u(j,1)=s(a(j))[q(a(1))
 50        +q(a(2))+--+q(a(j))]
 60    for k=2 until m do
 70     begin if k=m then k1=n else k1=k;
 80      for j=k1 until n do
 90       begin for l=k until j do
100        begin if a(1),a(2),--,a(l-1)の要素を
110          k-1 分割したときの最適解に部分集合
120          {a(l),a(l+1),--,a(j)} を追加した解
130          と、 a(1),--,a(j)をk-1 分割したとき
140          最適解の間に定理６が成立、   then
150           begin d(l,j)=s(a(j))[q(a(l)+q(a(l
```

```
160                +1)+--+q(a(j))],
170                v(l-k)=u(l-1,k-1)+d(l,k)
180              end
190          end
200          u(j,k)=min[v(0),v(1),--,v(j-k)]
210        u(j,k)に対応する最適解を記憶する。
220      end
230    end
240  end
```

20－30行は元の問題を変換問題にかえる。60－230行は順方向の最適性の原理による動的計画法の本体部分である。特に100-140行は層別化による目的関数の評価回数削減のためにある。

表1に示した数値例で目的関数uを最小にするような3分割を上記アルゴリズムに従ってもとめる。

u(1,1)=1, u(2,1)=6, u(3,1)=18, U(4,1)=40

u(5,1)=75, u(6,1)=114, u(7,1)=154, u(8,1)=192

u(2,2)=5 :{a(1)},{a(2)}

u(3,2)=min(16,15) :{a(1),a(2)},{a(3)}

表3.1 数値例

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
$s(a_i)$	1	2	3	4	5	6	7	8
$q(a_i)$	1	2	3	4	5	4	3	2

u(4,2)=min(37,34,34) :(a(1),a(2)),(a(3),a(4))

または (a(1),a(2),a(3)),(a(4))

u(5,2)=min(71,66,63,65) :(a(1),a(2),a(3)),

(a(4),a(5))

u(6,2)=min(109,102,96,94,99) :(a(1),a(2),a(3),

a(4)),(a(5),a(6))

u(7,2)=min(148,139,130,124,124,135) :(a(1),

a(2),a(3),a(4)),(a(5),a(6),a(7)) また

は (a(1),a(2),a(3),a(4),a(5)),(a(6),a(7))

u(8,2)=min(185,174,162,152,147,154,170) :(a(1)

a(2),a(3),a(4),a(5)),(a(6),a(7),a(8))

u(8,3)=min(173,159,146,135,134,140) :

(a(1),a(2),a(3),a(4)),(a(5),a(6)),

(a(7),a(8))

以上の計算によって、3分割のときの最適解は (a(1),a(2),a(3),a(4)),(a(5),a(6)),(a(7),a(8))で目的関数値は134である。なお、u(8,3)を求める式の下線部分は層別化によりこのアルゴリズムでは目的関数の評価から省略すべき部分であるが、あった方が理解しやすいので計算に加えた。

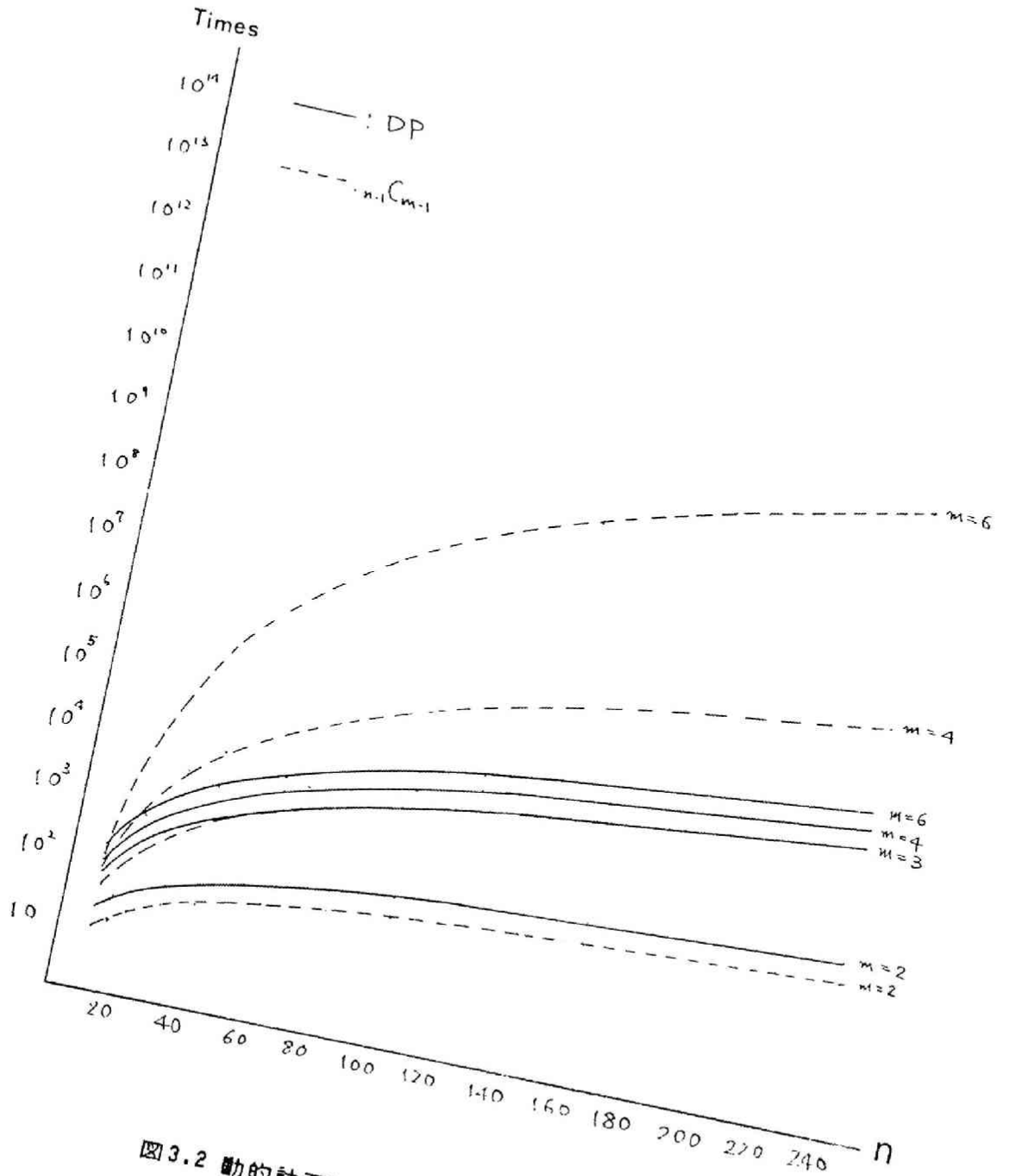


図3.2 動的計画法と組合せ法の計算回数の比較

この例では、目的関数u(j,k)の右辺の評価回数は、層別化したときは39回、しないときは42回である。一方、仕切を入れる組合せ法における目的関数uの評価回数はC(7,2)=21回である。しかし、m,nと評価回数の関係を図3.2で示したように、n≫m≫3のときには動的計画法が少ない評価回数ですむことが判る。

３．４．２．量子化可変長方式の最適化

量子化可変長マシンサイクル時間方式では、m種類のマシンサイクル時間を持つためには次式を満たすことが必要十分条件である。

$$\left.\begin{array}{l}(p-1)t(c)<t(1)\leq pt(c)\\(p+m-2)t(c)<t(n)\leq(p+m-1)t(c)\end{array}\right\} \quad [3.7]$$

ここで、t(c)は基本クロック周期、t(1)はマイクロ命令実行最短時間、t(n)はマイクロ命令実行最長時間及びpは正整数である。

[3.7]式より基本クロック周期の範囲を計算すると次式となる。

$$(t(n)-t(1))/m<t(c)<(t(n)-t(1))/(m-2) \quad [3.8]$$

[3.8]式の範囲にあって[3.7]式を満たすt(c)に対し

て次式のマシンサイクル時間T(j)及び実行頻度F(j)を求める。

$$T(j)=(p+(j-1))t(c) \quad (j=1,2,---,n)$$
$$F(j)=\sum_{i\in s(Tj)} f(i) \qquad [3.9]$$

ここで、s(T(j)) は (p+(j-1))t(c)<t(i)≦(p+(j-2))t(c) を満たす要素 i の集合である。

次に評価関数 u を計算する。

$$u=\sum_{j=1}^{m} F(j)T(j) \qquad [3.10]$$

評価関数が最小となるt(c)を求め、このt(c)に対するT(j)が最適なマシンサイクル時間である。

この問題を解く単純アルゴリズムを次に示す。

```
10   begin
20      t(c)の最小値と最大値を計算し、t(a),t(b) と
30      する。また uに仮の最小値をセットする。
40      for k=t(a) until t(b) do
50           begin p=1
60              while p*k<t(1) do p=p+1
70              for j=1 until m do
80                  begin T(j),F(j) を計算   end
```

```
 90              if T(m-1)<t(n)≦T(m) then
100                  begin 評価関数u'を計算する。
110                      if u'<u then
120                          begin u=u'とし、T(j)(j=1,-
130                              -,m) を記憶する。 end
140                  end
150          end
160  end.
```

３．５．おわりに

本章では、可変長マシンサイクル時間方式におけるマシンサイクル時間の最適化手法について述べた。

完全可変長マシンサイクル時間方式の最適化問題は、目的関数uが最小になるように集合Aを分割する問題であり、この最適解の性質を定理としてまとめた。これらの定理より、元の問題は各要素a(i)をs(a(i))の小さい順に並べかえた集合Aを目的関数uが最小となるようにm-1個の仕切を入れて分割する問題に変換できた。この変換により目的関数の評価回数がスターリング数S(n,m)から組み合わせ数C(n-1,m-1)に削減できた。

また、定理より、この変換問題を逐次決定法でo(m*n**2)回目的関数を評価して解くことができることを示した。さらに、この変換問題は、ネットワークの最短路問題に順方向の最適性の原理を適用した動的計画法と同一手法で最適解を求めることができ、このときの評価回数Tは(m-2)*n**2/2-m*(m-4)*n/2+m*(m**2-6*m+5)/6であることを示した。その上、最適解の範囲を限定することにより、T回以下の評価回数で最適解が求まることを示した。

次に、完全可変長マシンサイクル時間方式の最適なマシンサイクル時間を決定する問題を動的計画法で解くアルゴリズムと数値例を示し、また量子化可変長マシンサイクル時間方式の同様の問題を解く単純アルゴリズムを示した。

参考文献

1) Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: The Design and Analysis of Computer Algorithms, Addison-Wesley (1974)

2) Garey, M.R. and Johnson, D.S.: Computers and Intractability, W.H. Freeman and Company (1979)

3) C. ベルジュ著, 野崎昭弘訳: 組合わせ論の基礎, サイエンス社 (1973)

4) 岩根, 佐藤: ある種の割当て問題における解法について, 情報処理, Vol.23, No.2, pp187-193 (1982)

5) R.E. Bellman and S.E. Dreyfus: Applied Dynamic Programming, Princeton Univ. Press (1962)

6) 西川, 三宮, 茨木: 最適化, 岩波書店 (1982)

7) 岩根, 佐藤: ある分割問題の動的計画法による高速アルゴリズム, 情報処理論文誌 (投稿中)

第４章　マイクロ命令実行頻度に基づくタイミング設計

４．１．はじめに

可変長マシンサイクル時間方式には、完全可変長マシンサイクル時間方式と量子化可変長マシンサイクル時間方式があり、どちらの方式においても、マイクロ命令実行時間とマイクロ命令実行頻度が判れば、前章で議論した最適化技法によって最適なマシンサイクル時間を求めることができるが分かったが、これらを実現する方法並びに処理速度の向上の定量的な数値について解明されていない。

マイクロプログラム制御方式では、制御記憶から読み出されたマイクロ命令によって各回路の制御信号が作成されるので、処理速度を向上させるために、制御記憶の読み出しと既に読み出されたマイクロ命令の実行を並列処理する方式が多くの計算機に採用されているが[1]、可変長マシンサイクル時間方式に、この並列動作を採用したときのマイクロ命令実行時間と制御記憶のアクセス時間の関係は明らかでなかった。

また、可変長マシンサイクル時間方式のタイミング設

計時では、典型的なジョブを実行させた場合を仮定して最大公約数的なマイクロ命令実行時間分布に基づいて最適なマシンサイクル時間を決定している[2,3]。しかし、実際の使用環境は仮定したマイクロ命令実行時間分布をしておらず、この環境では最適なマシンサイクル時間とは言えない。それゆえ、使用環境に適するようにチューニングすることが必要である。

本章では、まず可変長マシンサイクル時間方式における制御記憶の読み出しとマイクロ命令実行の並列動作について議論する。そして完全可変長マシンサイクル時間方式と量子化可変長マシンサイクル方式の回路構成について述べる。次に、小型汎用計算機を設計してマイクロ命令実行時間分布を求め、この分布に基づいて、それぞれの可変長マシンサイクル時間方式について、マシンサイクル時間数毎に最適なマシンサイクル時間を計算して、マシンサイクル時間数、性能及び費用の観点から可変長マシンサイクル時間方式を議論する。また、使用環境に適するようにマシンサイクル時間をチューニングするとき、マシンサイクル時間数は固定してマシンサイクル時間のみを最適化するものとし、このときの性能改善率

について述べる。最後に完全可変長マシンサイクル時間方式及び量子化可変長マシンサイクル時間方式のチューニングに必要な回路構成について議論する。

４．２．可変長マシンサイクル時間方式の設計

４．２．１．並列処理方式

マイクロ命令の実行と次のマイクロ命令の読み出しを平行して行なう並列処理方式が性能向上のために多くの計算機に採用されている。図4.1と図4.2に固定長マシンサイクル時間方式と可変長マシンサイクル時間方式の並列処理のタイミングの概念を示す。図4.1及び図4.2において、T(m)及びT(m(i))はマシンサイクル時間、T(a)は制御記憶へのアクセス時間、T(e(i))はマイクロ命令iの実行時間であり、一般にT(a)は一定値をとり、T(e(i))はマイクロ命令の内容によって異なる。

並列処理方式におけるマシンサイクル時間は次式となる。

$$\left.\begin{array}{ll} T(a) \geq T(e(i)) \text{のとき} & T(m(i)) = T(a) \\ T(a) < T(e(i)) \text{のとき} & T(m(i)) = T(e(i)) \end{array}\right\} \quad [4.1]$$

また、直列処理方式はマイクロ命令の実行が完全に終了したあとで次のマイクロ命令の読み出しを行なう方式でマシンサイクル時間は次式となる。

$$T(m(i)) = T(a) + T(e(i)) \quad [4.2]$$

並列処理方式と直列処理方式と比較すると並列処理方

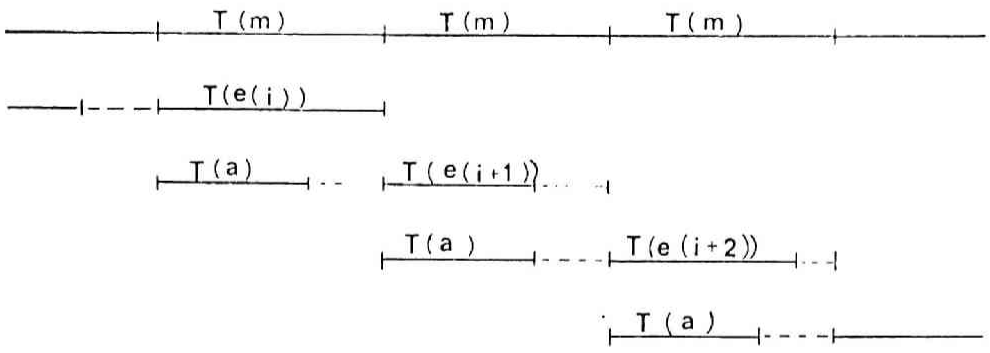


図4.1 固定長マシンサイクル時間方式

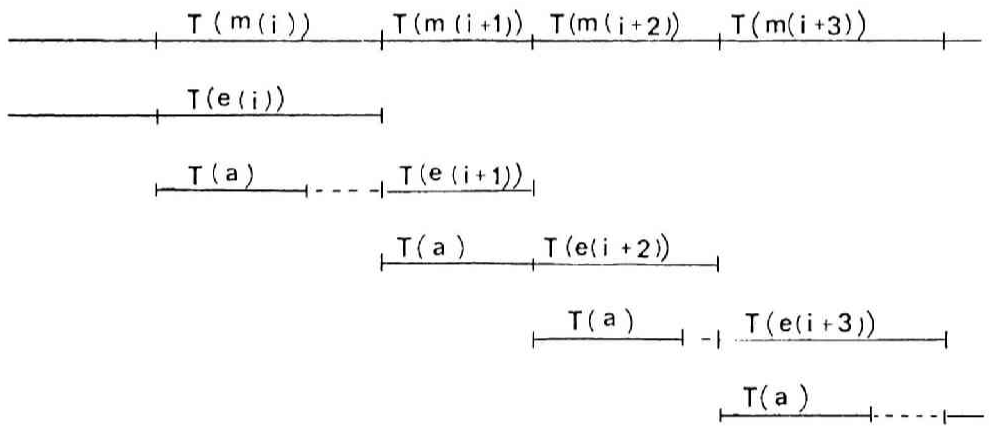


図4.2 可変長マシンサイクル時間方式

式が有利なのは次式が成立するときである[4]。

$$\sum_{i=1}^{m(1)} p(i)T(i) < \sum_{i=1}^{m(2)} T(a) \qquad [4,3]$$

ここでp(i)はマイクロ命令の読出しで誤まった分岐を予測する確率、T(i)はマイクロ命令の読出しで誤まった分岐を予測することによって生じる損失時間、T(a)は制御記憶へのアクセス時間、m(1)は命令を実行するのに必要なマイクロ命令数、m(2)は命令を実行するのに必要なマイクロプログラムのなかの分岐数である。

また、マイクロプログラムの1マイクロ命令前で分岐条件を指定するようにしてp(i)を 0にする方式も広く採用されている。

以上のことから、並列処理方式においてすべてのマイクロ命令に対してT(a)≧T(e(i))でないとき及び直列処理方式では可変長マシンサイクル方式が性能面で有利である。並列処理方式で可変長マシンサイクル方式をとるとき、T(e(i))<T(a)なるマイクロ命令実行時間T(e(i)) をT(a)とした度数分布を考えればよい。

4．2．2．完全可変長マシンサイクル時間方式

完全可変長マシンサイクル時間方式において、 n種類

のマイクロ命令実行時間t(i) (i=1,2,--,n) から m種類のマシンサイクル時間T(j)を第3章で述べた方法によって決定したとき、平均命令実行時間u(m)は次式で表わすことができる。

$$u(m) = h\left(\sum_{j=1}^{m} F(j)T(j)\right) \qquad [4.4]$$

ここでF(j)はマシンサイクル時間T(j)で動作するマイクロ命令の実行頻度である。

完全可変長マシンサイクル時間方式を実現するための制御記憶サブユニットの概念図を図4.3に、そのタイミング図を図4.4に示す。図4.3において、MCARはマイクロ命令アドレスレジスタ、MCIRはマイクロ命令レジスタ、CSは制御記憶、D(c),D(d),D(i) (i=1,2,--,m) は遅延線である。

水平形マイクロ命令形式では、一つのマイクロ命令で複数個のマイクロ操作を同時に平行して実行できるので暗黙のうちにマシンサイクル時間をマイクロ命令によって与えることはできない。それゆえマイクロ命令の 1フィールドをマシンサイクル時間指定用に割当てる必要がある。一方、垂直形マイクロ命令形式では、一つのマイクロ命令で一つのマイクロ操作を実行するので、マシン

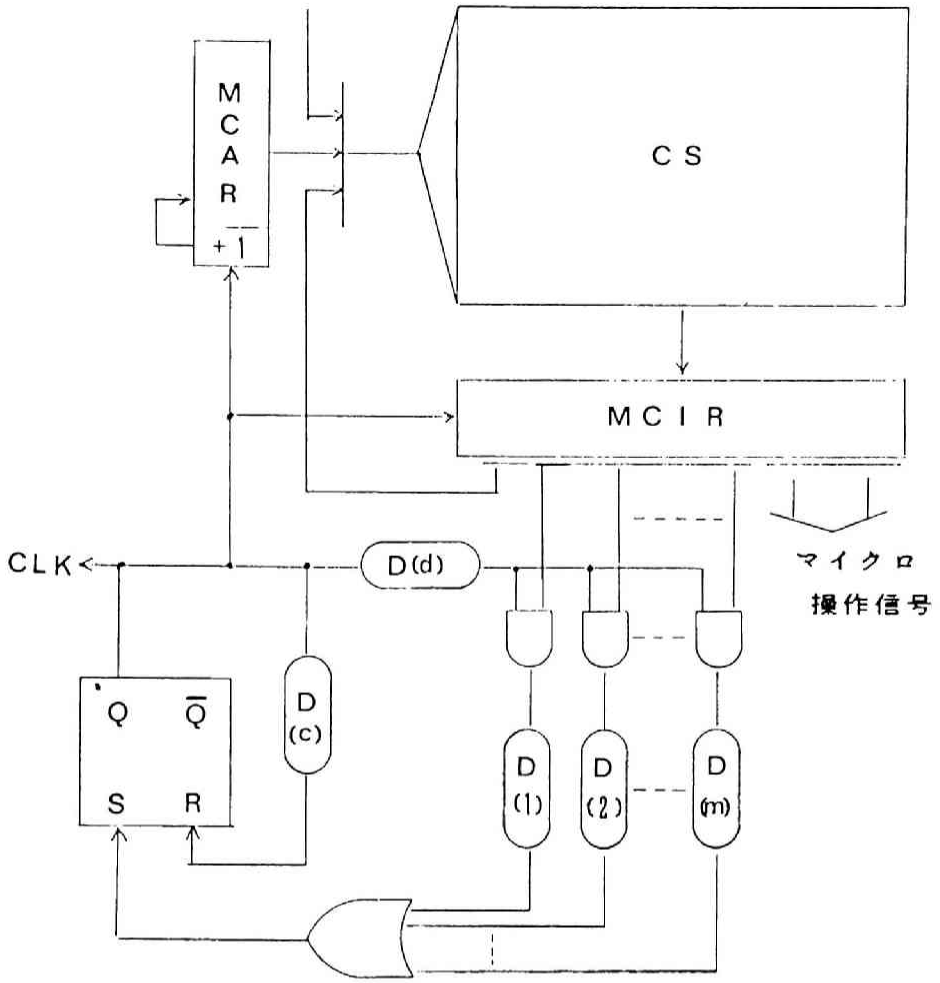


図4.3 完全可変長マシンサイクル時間方式ブロック図

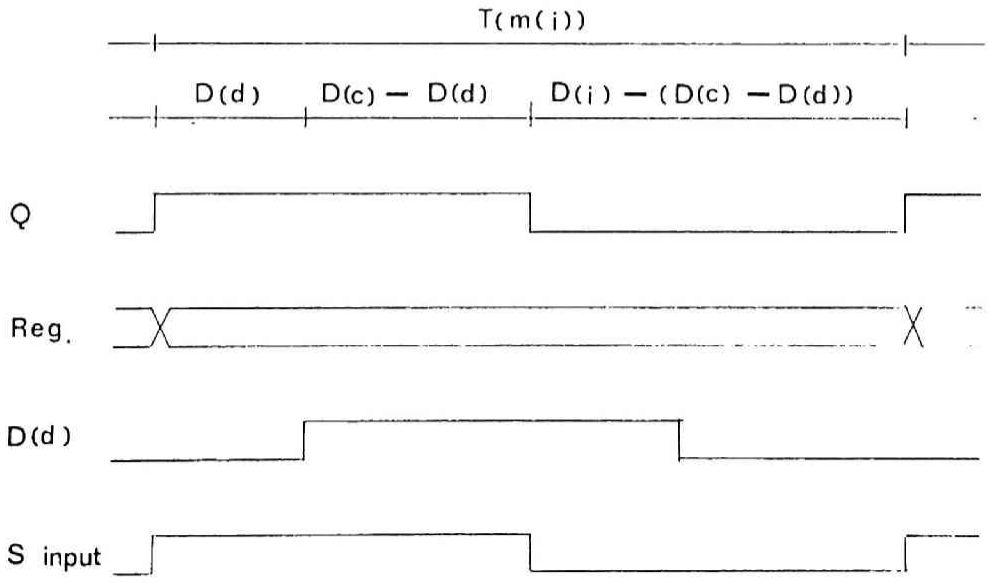


図4.4 完全可変長マシンサイクル時間方式タイミング図

サイクル時間とマイクロ命令とを一対一に対応させることができる。すると、垂直形マイクロ命令形式では、マシンサイクル時間指定用に特別なフィールドを設けても設けなくともどちらでもよい。

４．２．３．量子化可変長マシンサイクル時間方式

量子化可変長マシンサイクル時間方式において、マシンサイクル時間T(i)を基本クロックの連続した整数倍に制限すると、 m種類のマシンサイクル時間を持つためには、第２章及び第３章で議論したように次式を満足する必要がある。

$$\left.\begin{array}{l}(p-1)t(c)<t(1)\leq pt(c)\\ (p+m-2)t(c)<t(n)\leq(p+m-1)t(C)\end{array}\right\} \quad [4.5]$$

ここで、t(c)は基本クロック周期、t(1),t(n) はマイクロ命令実行時間の最小値と最大値、p は正の整数である。

すると、マシンサイクル時間T(j)は次式となる。

$$T(j) = (p+(j-1))t(c) \quad (j=1,2,--,m) \qquad [4.7]$$

またマシンサイクル時間T(j)で実行されるマイクロ命令M(j)の頻度F(j)は次式である。

$$F(j) = \sum_{i \in M(j)} f(i) \qquad [4.8]$$

ここで f(i) は T(j-1)<t(i)≦T(j)なるマイクロ命令実行時間 t(i)のマイクロ命令実行頻度である。

量子化可変長マシンサイクル時間方式における平均命令実行時間 u(m)は次式で与えられる。

$$u(m) = h\left(\sum_{j=1}^{m} F(j)T(j)\right) \qquad [4.9]$$

そこで、平均命令実行時間 u(m)が最小になる基本クロック周期を求めることにより、量子化可変長マシンサイクル時間方式の性能を最大にすることができる。

量子化可変長マシンサイクル時間方式を実現する制御記憶サブユニットの概念図を図4.5に、そのタイミング図を図4.6に示す。図4.5において、MCARはマイクロ命令アドレスレジスタ、MCIRはマイクロ命令レジスタ、CSは制御記憶、OSCは発振器、CNTRはカウンタを示す。また、マシンサイクル時間のためのマイクロ命令フィールドの設定に関しては完全可変長マシンサイクル方式と同様である。

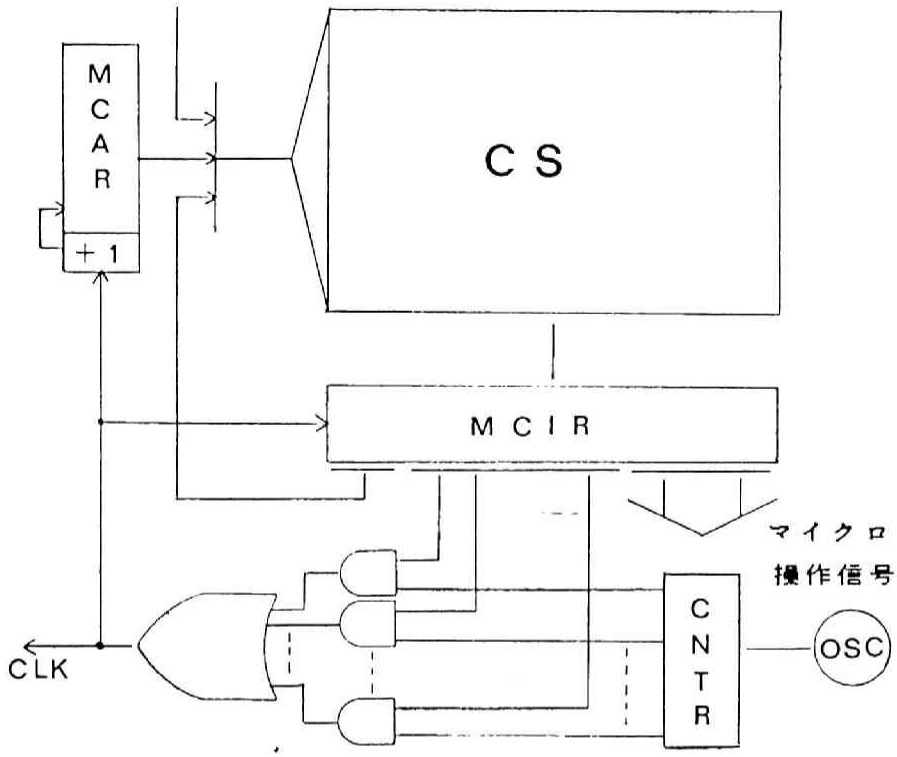


図4.5 量子化可変長マシンサイクル時間方式ブロック図

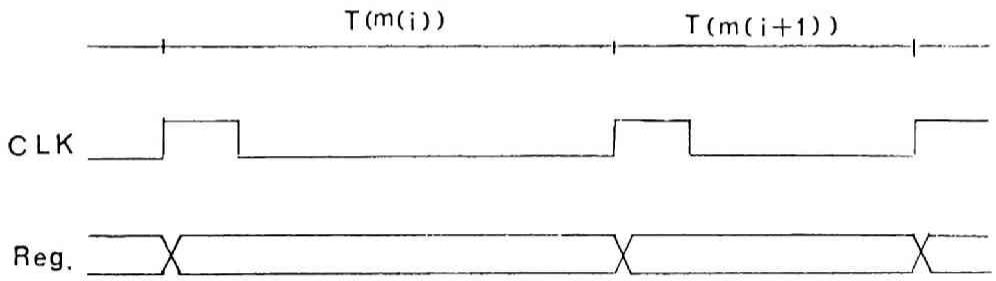


図4.6 量子化可変長マシンサイクル時間方式 タイミング図

４．３．方式の検討

４．３．１．実験用中央処理装置

図4.7のマイクロ命令フォーマット、120nsの制御記憶（CS）へのアクセス時間T(a)を持つ図4.8に示した中央処理装置（CPU）を使用して実験を行った[5,6]。

このCPUは24ビットを基本単位としており、その主構成要素は次の通りである。SPM,AR,BR,HRは演算汎用レジスタ、SARはSPMのアドレスレジスタであり、特にHRはシフトレジスタ及び外部系とのバッファレジスタとしても使用される。またMCD Selector,DVR Selectorは乗除算の高速化のための機能ユニットである。CTRは汎用カウンタ、IDはインジケータレジスタであり、CAR,MAR,BLR,BARは外部系へのアドレスを供給及び検査するレジスタである。CSは1K語の容量の制御記憶、CSARはCSアドレスレジスタ、MIRは同時平行処理のためのマイクロ命令バッファレジスタ、SQR,SVRは制御記憶アドレスの保持レジスタ、Branch Controlはマイクロプログラムの分岐を制御する機能ユニットである。

また、このCPUは、演算関係回路が6800ゲート、制御関係回路が2300ゲート、各種パネル関連回路を除く診断

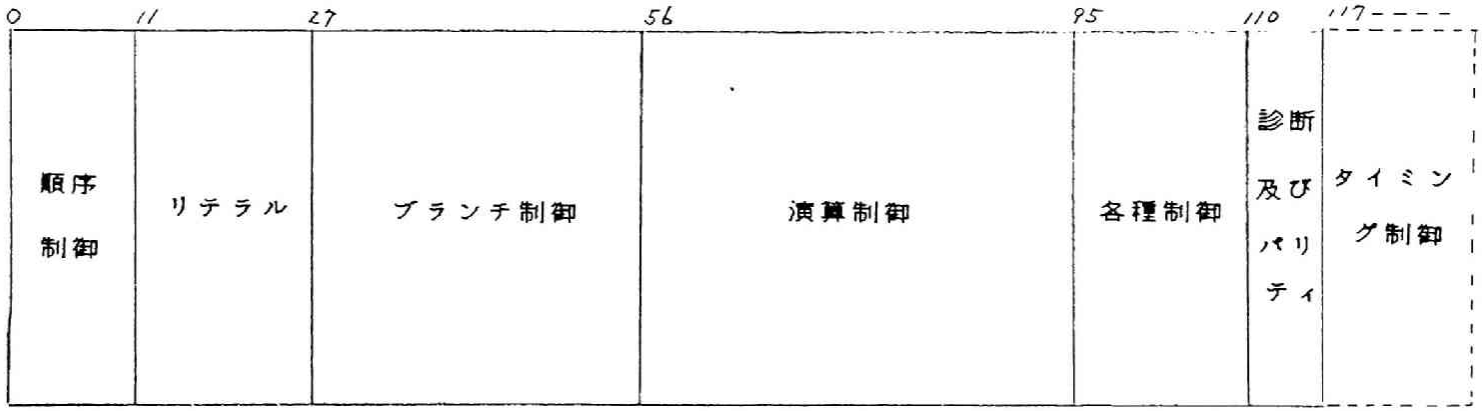


図4.7 マイクロ命令フォーマット

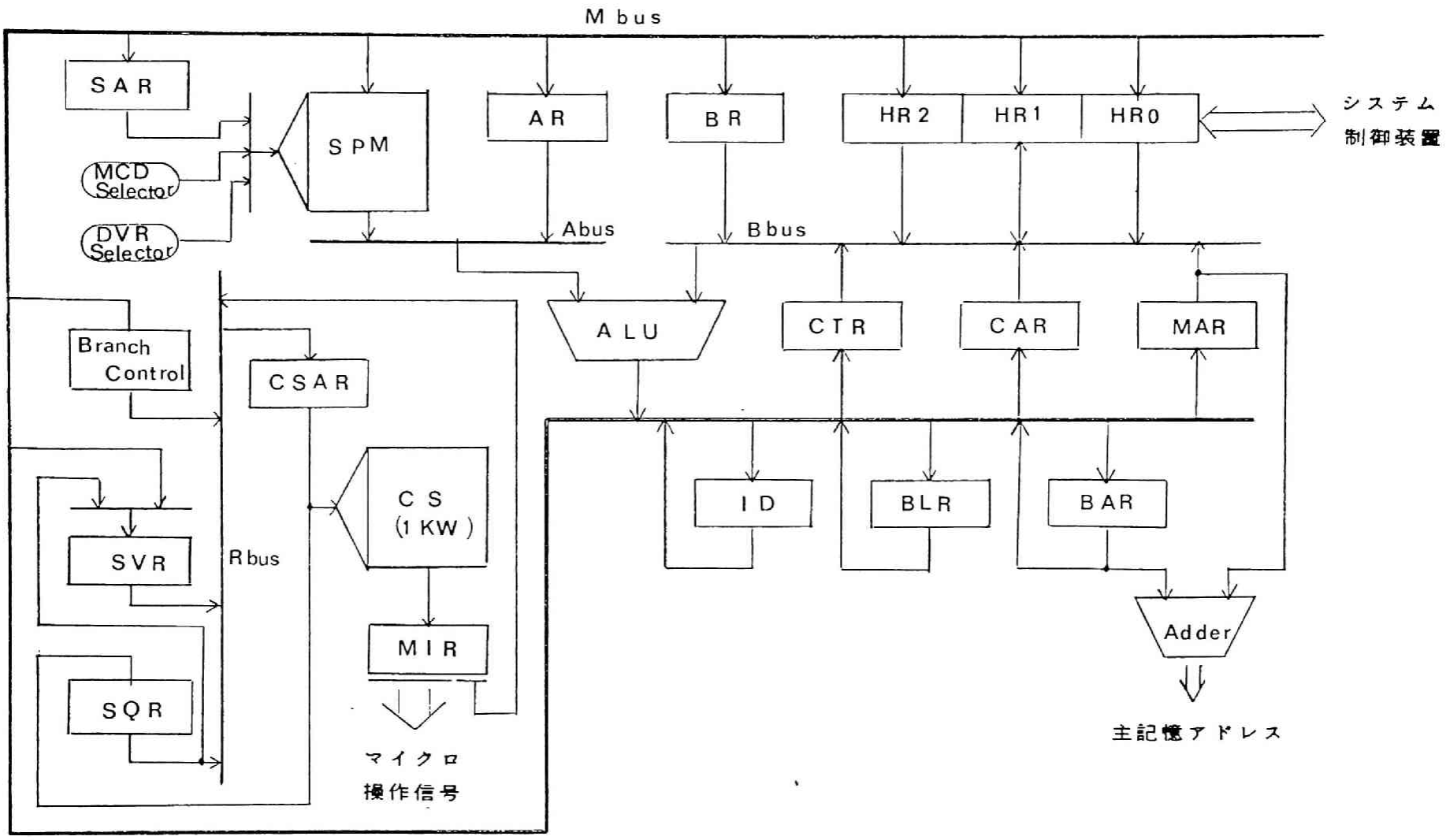


図4.8 CPU ブロック図

関係回路が1100ゲートで全体として約10000ゲートで構成されている。

このCPUの上で一般的なプログラムの幾つかを実行させ、すべてのマイクロ命令が含まれるようにして得られたマイクロ命令実行時間の分布を表4.1及び図4.9並びに図4.10に示す。図4.9は表4.1をそのままグラフにしたものであり、図4.10は 5ns毎に切り上げたものをグラフに表わしたものである。このデータの収集において、マイクロ命令実行時間は、技術者の計算による各マイクロ操作の実行時間を基にしてソフトウェアによって求められ、各マイクロ命令の実行頻度は、専用のマイクロプログラムシミュレータによって求めた。なお図4.9及び図4.10において、制御記憶のアクセス時間T(a)より短いマイクロ命令実行時間は120nsで代表されている。

図4.9で与えられた分布に対して、完全可変長マシンサイクル時間方式、量子化可変長マシンサイクル時間方式、マイクロ命令実行最短時間T(a)と最長時間t(n)の間に等間隔にマシンサイクル時間を設ける単純可変長マシンサイクル時間方式及び技術者の経験解による完全可変長マシンサイクル時間方式を採用したときのマシンサイ

表4.1 マイクロ命令実行時間分布

i	$t_{i(ns)}$	$f_{i(\%)}$	i	$t_{i(ns)}$	$f_{i(\%)}$	i	$t_{i(ns)}$	$f_{i(\%)}$
1	57	0.06	14	149	0.62	27	191	2.81
2	116	4.06	15	153	1.43	28	192	8.12
3	119	8.14	16	157	1.48	29	197	2.10
4	124	2.07	17	165	0.30	30	205	1.31
5	125	3.00	18	166	5.70	31	232	0.41
6	126	2.17	19	168	10.11	32	238	0.12
7	130	0.31	20	169	11.20	33	242	0.26
8	131	2.58	21	172	0.72	34	247	0.63
9	133	4.01	22	177	2.30	35	255	0.03
10	138	1.05	23	180	0.61	36	258	2.00
11	140	0.78	24	183	1.76	37	260	9.40
12	143	0.41	25	184	6.00	38	279	0.12
13	148	1.76	26	186	0.03	39	296	0.03

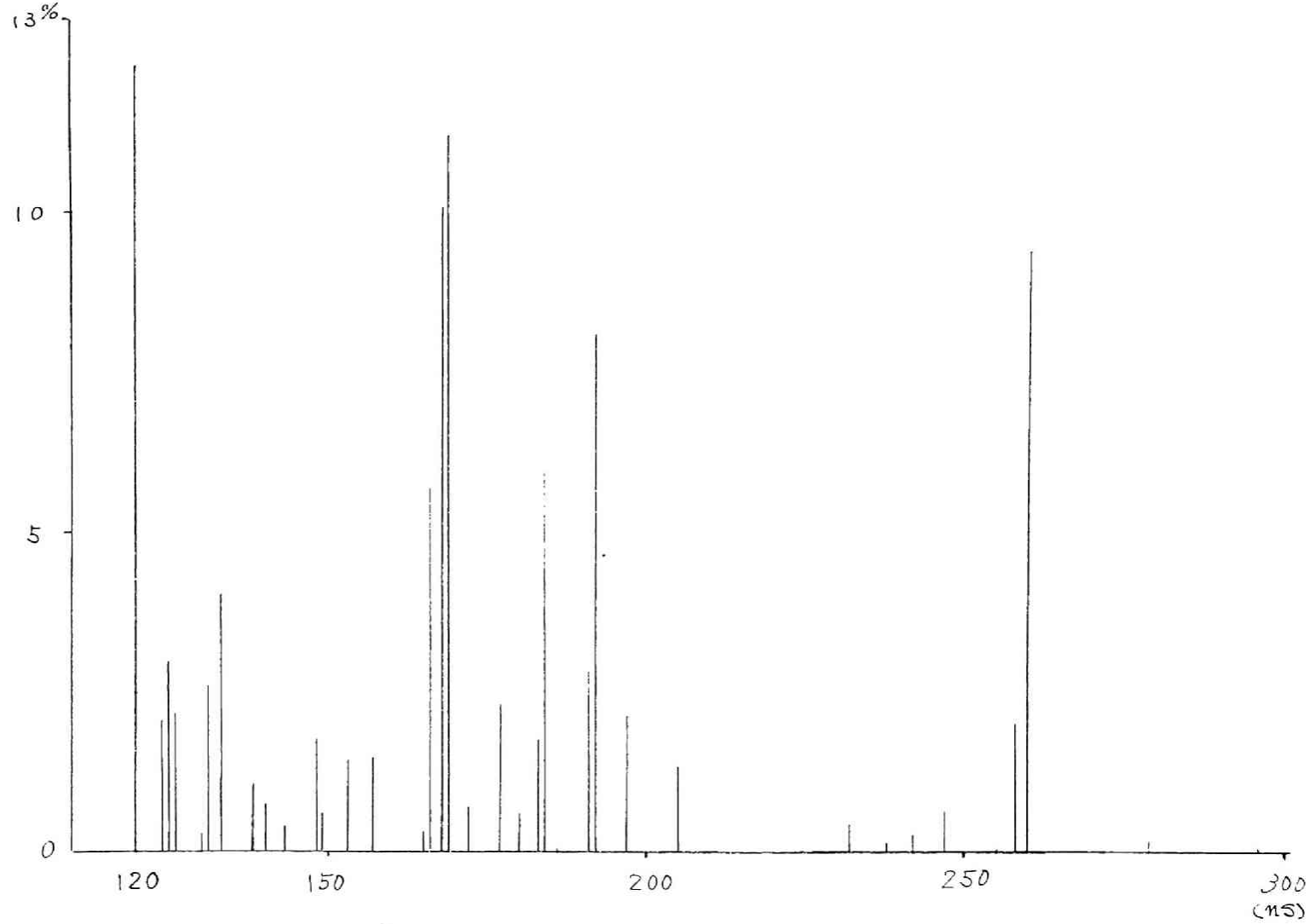


図4.9 マイクロ命令実行時間分布（1 ns単位）

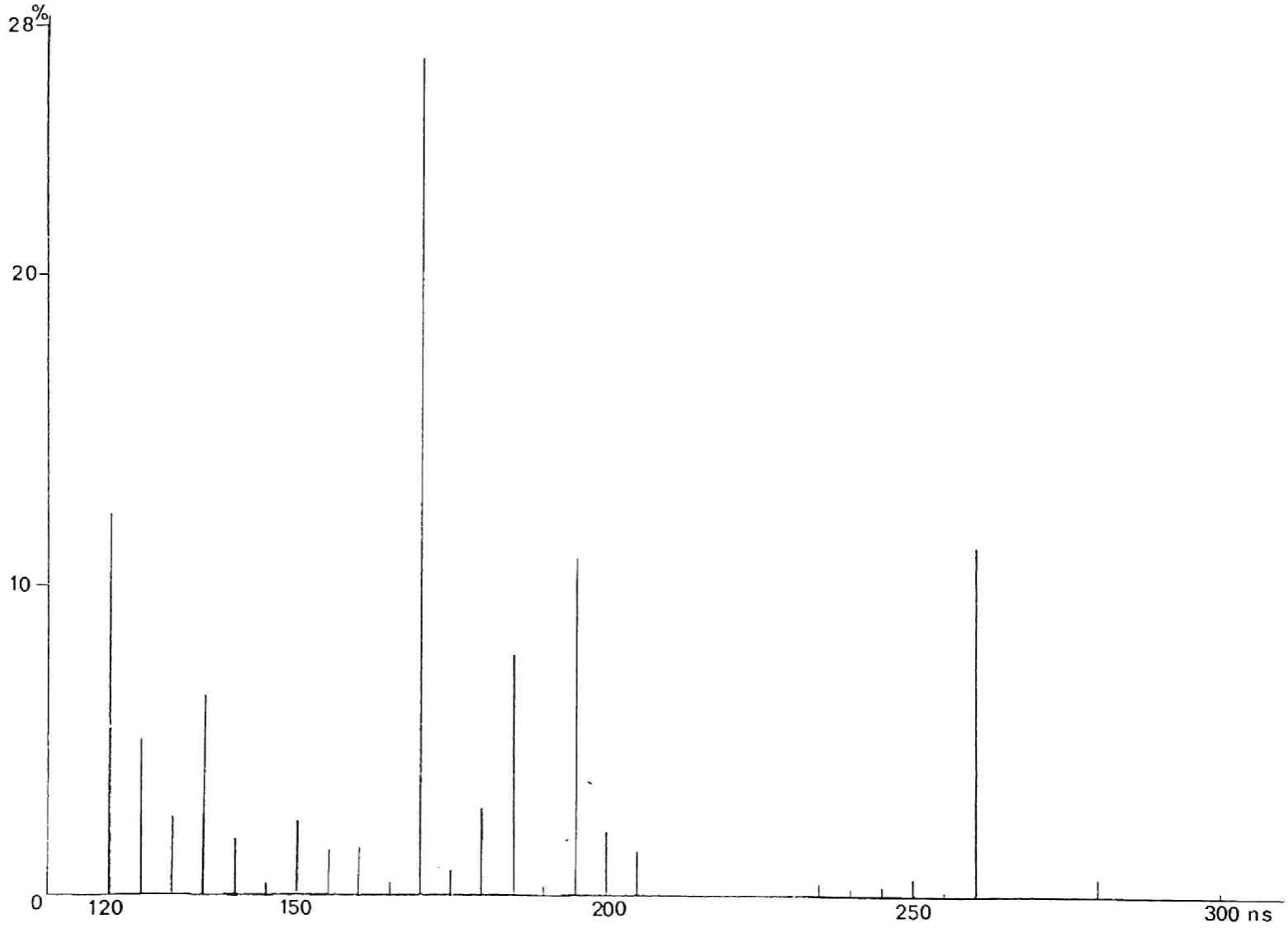


図4.10　マイクロ命令実行時間分布（５ns単位）

クル時間と性能向上比を表4.2に示す。また、完全可変長マシンサイクル時間方式と量子化可変長マシンサイクル時間方式1ns単位の最適解を各々 o,xでもって図4.11に示す。

4．3．2．比較検討

可変長マシンサイクル時間方式の設計において、マシンサイクル時間の種類数及びその時間を最適化する必要がある。そこで最適化のための評価関数として次式で示す性能価格比を考える。

$$z=(1/u(m))/(o+do) \qquad [4.10]$$

ここで、oは固定長マシンサイクル方式を採用したときのCPUの全ハードウェアの費用であり、doは可変長マシンサイクル時間方式を採用したときのハードウェア費用の増加分である。この増加費用は次式で与えられる。

$$do=a*m+b*w*[\log(m-1)+1] \qquad [4.11]$$

ここで、aは 1種類のマシンサイクルを実現するためのハードウェア費用、bは制御記憶のビットあたりの費用、wは制御記憶の語数であり、上式は水平形エンコードマイクロ命令形式を対象としており、垂直形マイクロ

表4.2 各種方式と性能向上比（その１）

m	1 ns完全可変長		5 ns完全可変長		1 ns量子化可変長		5 ns量子化可変長	
	T_i (ns)	μ_m/μ_1	T_i (ns)	μ_m/μ_1	T_i (ns)	μ_m/μ_1	T_i (ns)	μ_m/μ_1
1	296	1.000	300	1.014	296	1.000	300	1.014
2	192, 296	0.706	195, 300	0.717	198, 297	0.717	200, 300	0.724
3	133, 192, 296	0.654	170, 205, 300	0.662	174, 261, 348	0.700	174, 261, 348	0.702
4	133, 169, 197, 296	0.622	135, 170, 205, 300	0.631	132, 198, 264, 330	0.651	130, 195, 260, 325	0.652
5	133, 169, 197, 260 296	0.606	135, 170, 200, 260 300	0.612	132, 176, 220, 264, 308	0.637	132, 176, 220, 264, 308	0.641
6	133, 169, 192, 205 260, 296	0.601	120, 135, 170, 200 260, 300	0.605	136, 170, 204, 238, 272, 306	0.619	136, 170, 204, 238 272, 306	0.620
7	120, 133, 169, 192, 205, 260, 296	0.595	120, 135, 170, 185 205, 260, 300	0.600	145, 174, 203, 232, 261, 290, 319	0.624	145, 174, 203, 232 261, 290, 319	0.625
8	120, 133, 153, 169, 192, 205, 260, 296	0.592	120, 135, 170, 185 195, 205, 260, 300	0.596	125, 150, 175, 200, 225, 250, 275, 300	0.615	125, 150, 175, 200, 225, 250, 275, 300	0.615
16	120, 126, 133, 140, 149, 157, 166, 169, 177, 184, 192, 197, 205, 247, 260, 296	0.583	120, 125, 130, 135 140, 150, 160, 170 180, 185, 195, 200 205, 250, 260, 300	0.587	120, 132, 144, 156, 168, 180, 192, 204, 216, 228, 240, 252, 264, 276, 288, 300	0.596	120, 132, 144, 156, 168, 180, 192, 204, 216, 228, 240, 252, 264, 276, 288, 300	0.608
26	120, 125, 126, 131 133, 140, 143, 149 153, 157, 166, 168 169, 172, 177, 180 184, 191, 192, 197 205, 232, 247, 258 260, 296	0.581	120, 125, 130, 135 140, 145, 150, 155 160, 165, 170, 175 180, 185, 190, 195 200, 205, 235, 240 245, 250, 255, 260 280, 300	0.587	126, 133, 140, 147 154, 161, 168, 175 182, 189, 196, 203 210, 217, 224, 231 238, 245, 252, 259 266, 273, 280, 287 294, 301	0.594	126, 133, 140, 147 154, 161, 168, 175 182, 189, 196, 203 210, 217, 224, 231 238, 245, 252, 259 266, 273, 280, 287 294, 301	0.600

表4.2 各種方式と性能向上比（その２）

m	1 ns単純可変長		5 ns単純可変長		1 ns技術者		5 ns技術者	
	T_i (ns)	μ_m/μ_1	T_i (ns)	μ_m/μ_1	T_i (ns)	μ_m/μ_1	T_i (ns)	μ_m/μ_1
1	296	1.000	300	1.014	296	1.000	300	1.014
2	208, 296	0.741	210, 300	0.749	192, 296	0.706	170, 300	0.745
3	179, 237, 296	0.700	180, 240, 300	0.705	133, 192, 296	0.654	170, 205, 300	0.662
4	164, 208, 252, 296	0.689	165, 210, 255, 300	0.695	133, 169, 192, 296	0.627	135, 170, 205, 300	0.631
5	155, 190, 226, 261 296	0.652	156, 192, 228, 264, 300	0.658	133, 169, 205, 260 296	0.610	135, 170, 205, 260, 300	0.613
6	149, 179, 208, 237 267, 296	0.634	150, 180, 210, 240, 270, 300	0.638	133, 169, 192, 205 260, 296	0.601	120, 135, 170, 205, 260, 300	0.607
7	145, 170, 195, 221 246, 271, 296	0.619	146, 171, 197, 223, 249, 274, 300	0.624	120, 133, 169, 192 205, 260, 296	0.595	120, 135, 170, 185, 205, 260, 300	0.599
8	142, 164, 186, 208, 230, 252, 274, 296	0.631	143, 165, 188, 210 233, 255, 278, 300	0.637	120, 133, 169, 184 192, 205, 260, 296	0.592	120, 135, 170, 185, 195, 205, 260, 300	0.596
16	131, 142, 153, 164 175, 186, 197, 208 219, 230, 241, 252 263, 274, 285, 296	0.601	131, 143, 154, 165, 176, 188, 199, 210, 221, 233, 244, 255, 266, 278, 289, 300	0.607	———	—	———	—
26	127, 134, 140, 147 154, 161, 167, 174 181, 188, 194, 201 208, 215, 222, 228 235, 242, 249, 255 262, 269, 276, 282 289, 296	0.593	127, 134, 141, 148, 155, 162, 168, 175, 182, 189, 196, 203, 210, 217, 224, 231, 238, 245, 252, 258, 265, 272, 279, 286, 293, 300	0.601	———	—	———	—

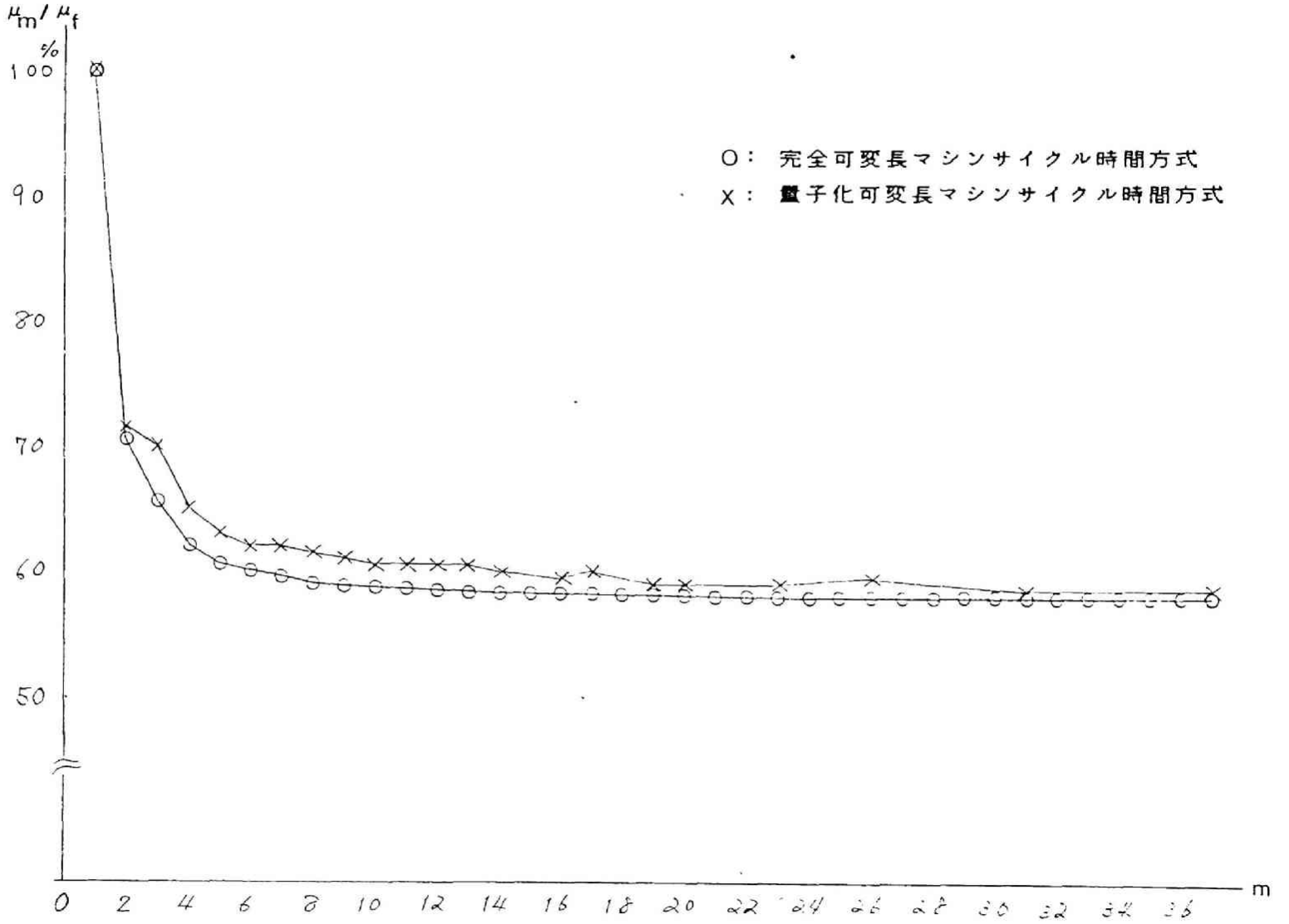


図4.11 マシンサイクル時間の種類数と性能

命令形式では第２項は 0である。

最適なマシンサイクル時間の種類を決定することは、評価関数zが最大となるmとT(1),T(2),--T(m)を選ぶことである。図4.9の分布を持つジョブ環境でのマシンサイクル時間の種類を決定するとき、むやみに種類を多くしても無意味であることが表4.2及び図4.11から判明する。すなわち、マシンサイクル時間の種類mが 4種類まで平均命令実行時間比(u(m)/u(f))は種類 mの増加とともに急激に小さくなり、 8種類までこの時間比は種類mの増加とともになだらかになり、 9種類以上のときいくら種類 mが増えてもこの時間比は殆んど変化しない。それゆえ、マシンサイクル時間の 9種類以上のとき評価関数値は減少することが明らかであるので、 9種類以下の場合について検討する。

図4.9の分布の環境では、可変長マシンサイクル時間方式の平均命令実行時間は、マシンサイクル時間の種類数mが多くなるのに従ってT(f)=296nsの固定長マシンサイクル時間方式の平均命令実行時間の約60%に近ずき、また分布を1ns 単位から5ns 単位に圧縮したとき 1ns単位の平均命令実行時間の 1%弱長くなるだけで大差がな

いことが分かる。

また量子化可変長マシンサイクル時間方式の平均命令実行時間は完全可変長マシンサイクル時間方式より 1.6～7.0 %長くなり、単純可変長マシンサイクル時間方式の平均命令実行時間は完全可変長マシンサイクル時間方式より 4.0～10.7%長くなる。

完全可変長マシンサイクル時間方式では、一般に次式が成立する。

$$u(m-1) \rangle u(m) \qquad [4.12]$$

しかし、他の可変長マシンサイクル時間方式では[4.12]式が必ずしも成立しない。例えば、量子化可変長マシンサイクル時間方式では$u(6) \langle u(7)$、単純可変長マシンサイクル時間方式では$u(7) \langle u(8)$であることが表4.2から明らかである。また量子化可変長マシンサイクル時間方式ではマシンサイクルの種類m=15,18,21,22等のように解が存在しない場合がある。

それゆえ量子化可変長マシンサイクル時間方式及び単純可変長マシンサイクル時間方式を採用するときはマシンサイクル時間の種類mに注意する必要がある。

さて、4.11式において一般に$a*m \langle\langle b*w*[\log(m-1)+1]$

であり、また制御記憶用メモリ素子のビットあたりの単価は非常に安くなっているので、実験機では完全可変長マシンサイクル時間方式を採用し、5nsに圧縮した分布を用いて 8種類のマシンサイクル時間を持たせた。その結果、3kビットの制御記憶と 8個の遅延線及び数十ゲートの増加によって、平均命令実行時間はT(f)=296nsの固定長マシンサイクル時間方式の場合の0.596倍に短縮できた。このとき、ギブソンミックスによる平均命令実行時間は 2.5ns(0.4MIPS)であった。

一般に、マイクロ命令実行時間T(e(i))が大きくばらつき、制御記憶へのアクセス時間T(a)とMax T(e(i))との差が大きいほど、可変長マシンサイクル方式は有効である。また、命令の先回り処理にパイプライン処理を採用[7]しているCPUにおいても、パイプラインの次のステージに命令語を送るために必要な最長時間をT(p)とするとMax(T(p),T(a))とMaxT(e(i))の差が大きいほど可変長マシンサイクル方式は有効である。

４．４．チューニングによる性能改善

４．４．１．チューンアップの性能予測

CPU の設計時では、典型的な幾つかのジョブを実行させた場合を仮定して最大公約数的なマイクロ命令実行時間分布に基ずいて、最適なマシンサイクル時間の種類数とその時間を決定した。しかし、CPU の使用される環境は必ずしも仮定したマイクロ命令実行時間分布をしておらず、その環境では最適なマシンサイクル時間とは言えない。そこで使用環境に適したようにマシンサイクル時間をチューンアップする必要があるが、ここではマシンサイクル時間の種類数は固定してその時間のみを適正化することを考察する。

さて、マシンサイクル時間の種類数をm，チューンアップ前の平均命令実行時間をu(m)，チューンアップ後の平均命令実行時間をu'(m) とすると、チューンアップによる改善率は次式で定義できる。

$$r = (u(m)-u'(m))/u(m) \qquad [4.13]$$

この定義に基づいて、まず完全可変長マシンサイクル時間方式におけるチューンアップによる性能改善率を算出する。設計時の最適マシンサイクル時間をT(1)，T(2)，

--,T(m) とし、このマシンサイクル時間にとって最悪なマイクロ命令実行時間分布を考える。すなわち、T(j+1)-T(j) が最大であるj に対するT(j)をT(k)とし、t=T(k)+1のとき頻度f(t)=1,t≠ T(k)+1のとき f(t)=0をとる分布を考える。この分布に対するチューンアップ前の平均命令実行時間u(m)は、u(m)=hT(k+1)であり、チューンアップ後は、u'(m)=h(T(k)+1) となる。但し、 hは定数である。すると性能改善率r は次式となる。

r=(T(k+1)-T(k)-1)/T(k+1) [4.14]

マイクロ命令実行時間の最小値をt(1), 最大値をt(n)とすると、t(n)=qt(1) (q)1) とおける。すると、性能改善率の最大値r(max)は次式となる。

r(max)((q-1)t(1)-1)/qt(1)

(T(k)=t(1),T(k+1)=t(n)) [4.15]

すると性能改善率r は次式で表すことができる。

0(r(1-1/q (t(1))0) [4.16]

次に量子化可変長マシンサイクル時間方式のチューンアップによる性能改善率を計算する。完全可変長マシンサイクル時間方式と同様に、T(1),T(2),--,T(m) を設計時の最適マシンサイクル時間とする。

$$T(k)=(p+k-1)t(c) \quad (p\geq 1, k=1,2,\cdots,m) \qquad [4.17]$$

ここで t(c) は基本クロックの周期である。このマシンサイクル時間にとって最悪分布を考える。すなわち t=T(k)+1 のとき頻度 f(t)=1, t≠T(k)+1のとき f(t)=0である分布を考える。するとこの分布に対するチューンアップ前の平均命令実行時間 u(m)は、u(m)=hT(k+1)であり、チューンアップ後では u'(m)=h(T(k)+1) となる。但し h は定数である。すると性能改善率 r は次式となる。

$$r=(T(k+1)-T(k)-1)/T(k+1) \qquad [4.18]$$

すると、4.17式より性能改善率の最大値 r(max)は次式となる。

$$r(max)=(1-1/t(c))/(p+k) \qquad [4.19]$$

また、マイクロ命令実行時間の最小値を t(1), 最大値を t(n)とすると、次式が成り立つ。

$$t(1)/t(c)<p<t(1)/t(c)+1, \quad p\geq 1 \qquad [4.20]$$

$$\left.\begin{array}{l}(m-2)/(q-1)<t(1)/t(c)<m/(q-1),\\ \qquad\qquad t(n)=qt(1), q>1\end{array}\right\} \qquad [4.21]$$

これらの式より、

(m-2)/(q-1)>1 のとき、

$$(m-2)/(q-1)<p<m/(q-1)+1 \qquad [4.22]$$

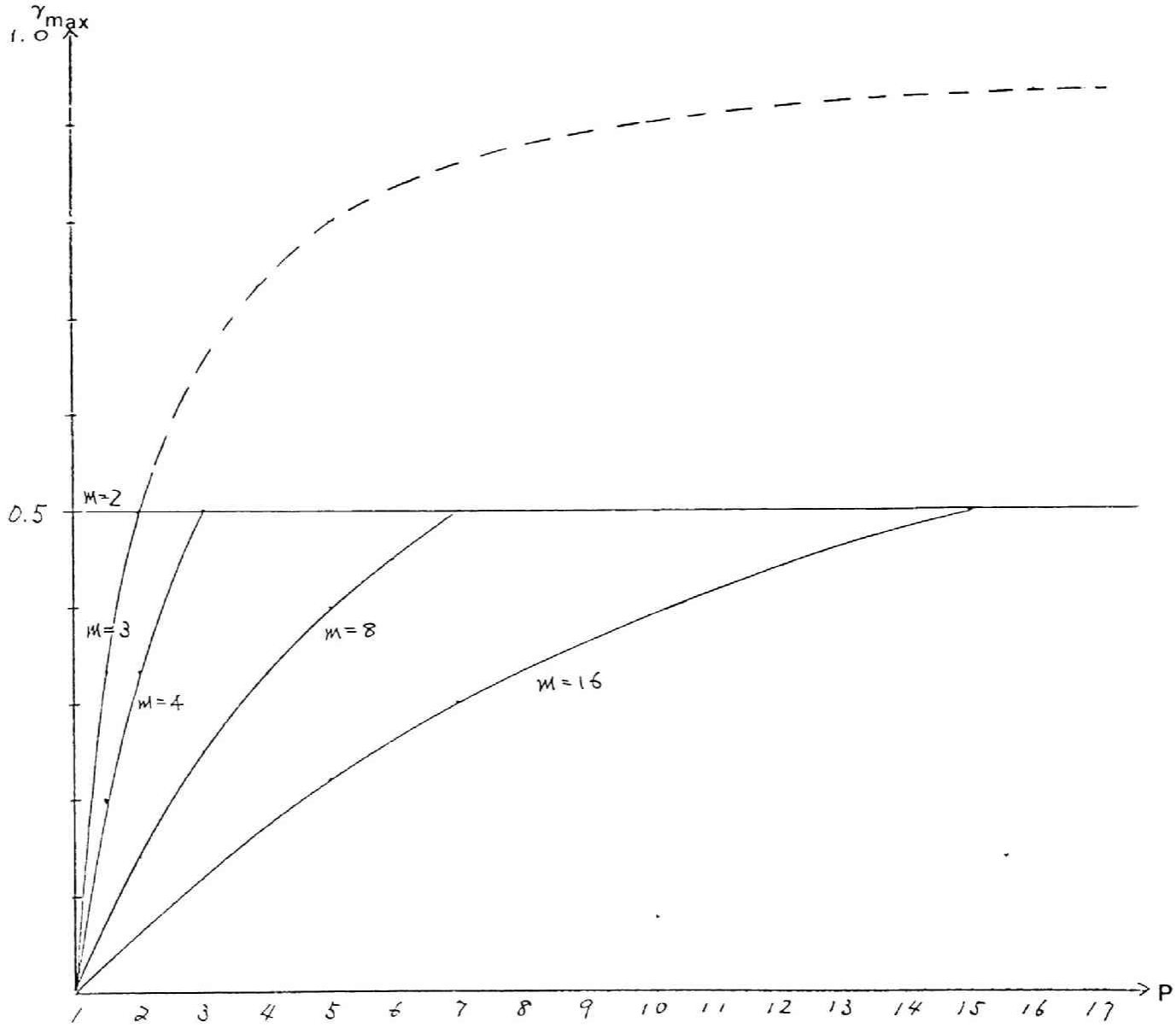


図4.12　マイクロ命令実行時間のばらつきと性能改善率

表4.3 シミュレーションデータ

t_i	f_i		t_i	f_i		t_i	f_i	
	例 I	例 II		例 I	例 II		例 I	例 II
ns	%	%	ns	%	%	ns	%	%
52	0.10	8.12	149	2.60	1.76	191	3.00	0.03
116	0.20	2.10	153	3.00	0.06	192	2.60	0.62
119	0.20	1.31	157	3.40	4.06	197	2.20	1.43
124	0.60	0.41	165	4.00	8.14	205	1.80	1.48
125	0.70	0.12	166	4.60	2.07	232	1.60	0.30
126	0.80	0.26	168	5.20	3.00	238	1.40	5.7
130	0.90	0.63	169	8.50	2.17	242	1.20	10.11
131	1.00	0.03	172	12.00	0.31	247	1.00	11.20
133	1.20	2.00	177	8.50	2.58	255	0.90	0.72
138	1.40	9.40	180	5.20	4.01	258	0.80	2.30
140	1.60	0.12	183	4.60	1.05	260	0.70	0.61
143	1.80	0.03	184	4.00	0.78	279	0.60	1.76
148	2.20	2.81	186	3.40	0.41	296	0.50	6.00

表4.4 完全可変長マシンサイクル時間方式
のチューンアップによる改善率

m	例 I			例 II		
	T_i (tune up 前)	T_i (tune up 後)	γ	T_i (tune up 前)	T_i (tune up 後)	γ
2	192, 296	192, 296	0	192, 296	180, 296	0.016
3	133, 192, 296	172, 197, 296	0.042	133, 192, 296	169, 247, 296	0.052
4	133, 169, 197, 296	172, 192, 247, 296	0.018	133, 169, 197, 296	138, 180, 247, 296	0.058
8	120, 133, 153, 169 192, 205, 260, 296	140, 157, 172, 184 192, 205, 260, 296	0.019	120, 133, 153, 169 192, 205, 260, 296	120, 138, 169, 184 205, 247, 260, 296	0.023

表4.5 量子化可変長マシンサイクル時間方式
のチューンアップによる改善率

m	例 I			例 II		
	T_i (tune up 前)	T_i (tune up 後)	γ	T_i (tune up 前)	T_i (tune up 後)	γ
2	198, 297	198, 297	0	198, 297	198, 297	0
3	174, 261, 348	186, 279, 372	0.046	174, 261, 348	166, 249, 332	0.012
4	132, 198, 264, 330	124, 186, 248, 310	0.019	132, 198, 264, 330	150, 200, 250, 300	0.045
8	125, 150, 175, 200 225, 250, 275, 300	138, 161, 184, 207 230, 253, 276, 299	0.001	125, 150, 175, 200 225, 250, 275, 300	125, 150, 175, 200 225, 250, 275, 300	0

(m-2)/(q-1)<1 のとき、 1<p<m/(q-1)+1 [4.23]

故に性能改善率rは次式となる。

m>q+1 のとき 0<r<(q-1)/(m+q-3) (t(o))>0) [4.24]

m<q+1 のとき 0<r<1/2 (t(o))>0) [4.25]

以上の結果をまとめると、完全可変長マシンサイクル時間方式の性能改善率は図4.12の破線となり、量子化可変長マシンサイクル時間方式では図4.12の実線となる。さらに表4.1に示した設計時のマイクロ命令実行時間分布が表4.3の例1,2のように変ったとき、これをチューンアップすると表4.4及び表4.5のようになり、これらの例では最大 6%程度性能が改善される。

4.4.2. 自動チューニング法

個々の使用環境に最適なチューンアップを施すことによって性能が改善できることが判明したので、ここでは自動的にチューンアップする技法について考察する。

チューンアップは、モニタリング、解析、再構成の3段階からなり、モニタリングはマイクロ命令実行番地を記録し、解析は設計時に算出したマイクロ命令実行時間とモニタリングから得られた頻度より新マシンサイクル

時間を決定し、そして再構成はマシンサイクル時間データを制御記憶並びにタイミング指定レジスタにロードすることである。

図4.13に完全可変長マシンサイクル時間方式、図4.14に量子化可変長マシンサイクル時間方式の自動チューニングを実現するための概念図を示す。図4.13及び図4.14において、少なくとも制御記憶CSのタイミングフィールドは書き込み可能であり、かつマイクロ命令実行番地がなんらかの形で外部へ転送できる。図4.13において、PD(i)はプログラマブル遅延線であり、内蔵マルチプレクサの選択によって適切な遅延時間を選ぶことができ、また TDRはマルチプレクサ選択信号を与えるためのタイミング指定レジスタであり、外部からロード可能な構造である。図4.14において、PLL は位相同期ループ、MR,NRは定数M,N を貯えるレジスタであり、 PLLの入力周波数をf(i)、出力周波数をf(o)とすると、f(o)=Nf(i)/Mである。また PCOUNTER はプログラマブルカウンタ、KRはカウント数を貯えるカウンタであり、MR,NR,KRは外部からロード可能である。

モニタリングによってマイクロ命令実行頻度を求め、

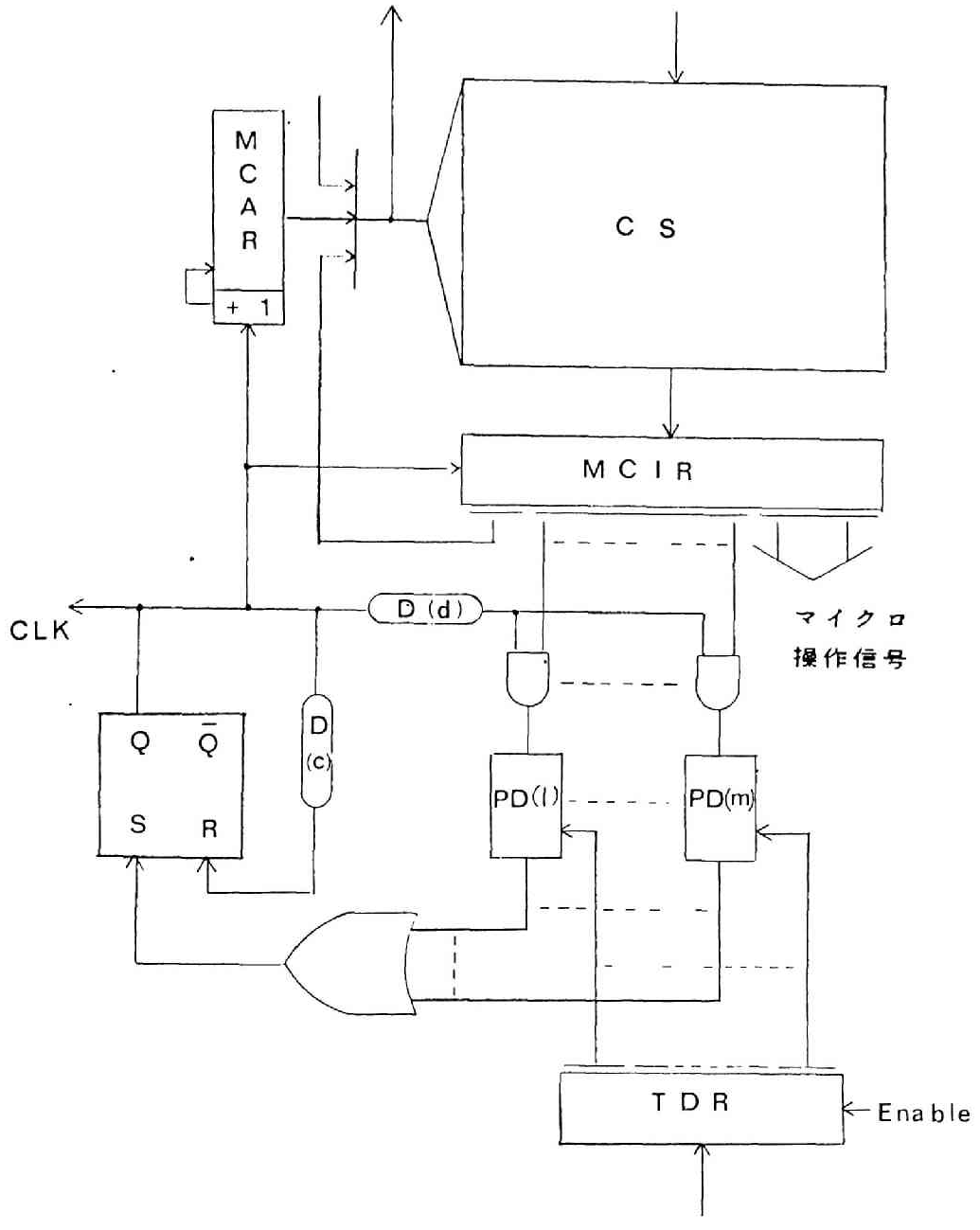


図4.13　完全可変長マシンサイクル時間方式の自動チューンアップ機構

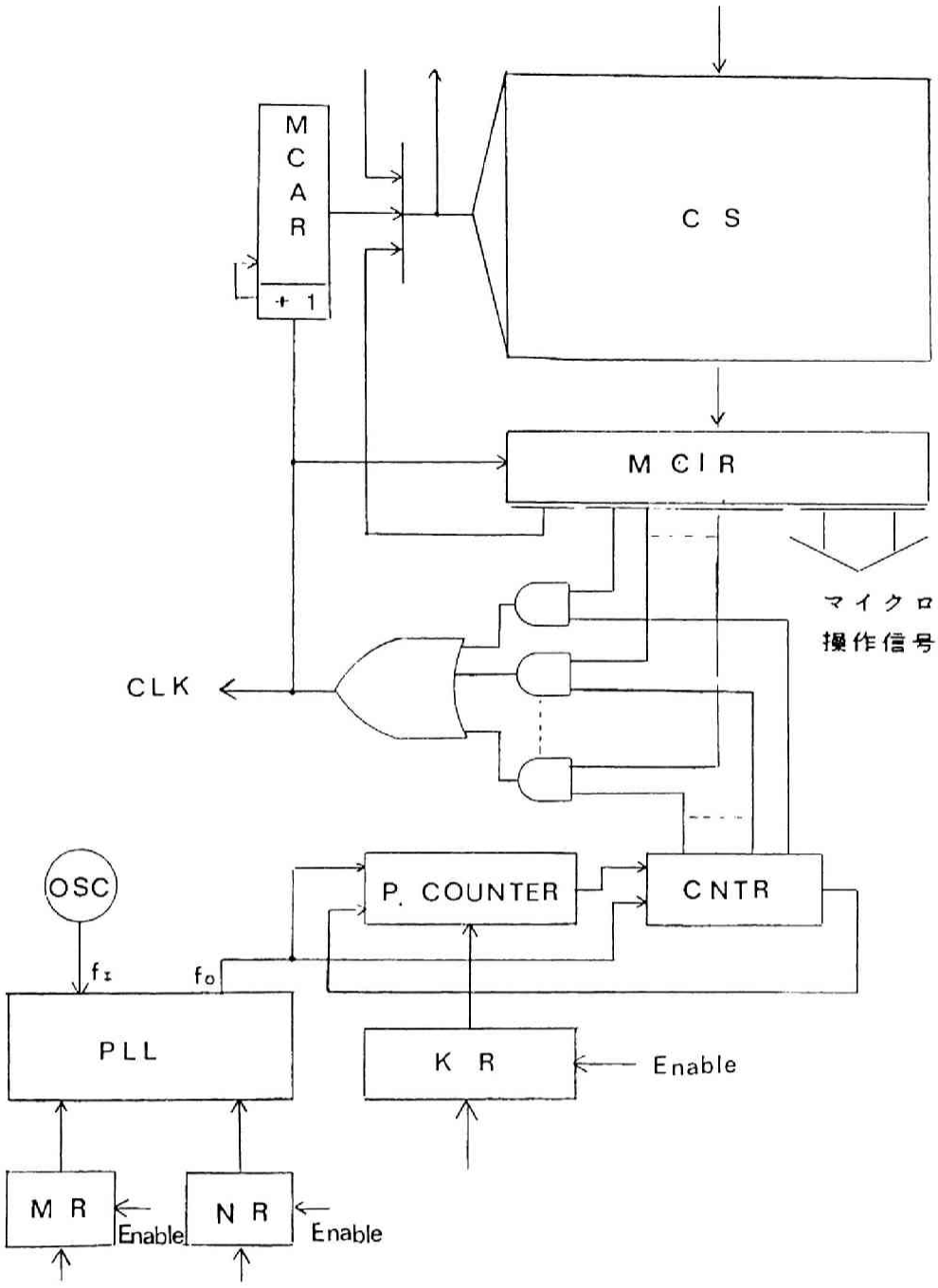


図4.14　量子化可変長マシンサイクル時間方式 の自動チューンアップ機構

この分布に基づいて 3章で示したアルゴリズムによってマシンサイクル時間を算出し、スタートアップで制御記憶のタイミングフィールド、MR,NR,KRへタイミングデータをロードすることによって自動チューニングが可能である。

４．５．おわりに

本章では、制御記憶の読み出しとマイクロ命令の同時平行動作を伴った可変長マシンサイクル時間方式の設計時における最適化と実際の使用環境における最適化について論じた。

同時平行処理方式において、制御記憶のアクセス時間がマイクロ命令実行時間より短い時は、可変長マシンサイクル時間方式が性能面で有利である。120ns の制御記憶のアクセス時間を持ち、マイクロ命令実行時間が52ns～296ns であり、制御記憶の同時平行動作を行うCPU のマシンサイクル時間の最適化を行った。

まず、典型的なジョブのマイクロ命令実行時間分布による設計時の最適化について考察した。実験例では、可変長マシンサイクル時間方式の平均命令実行時間は、マシンサイクル時間数m が多くなるのに従って固定長マシンサイクル時間方式の60%に近ずき、マシンサイクル時間の種類をむやみに多くしても、この時間比は飽和して殆んど変化しないことが判った。また、完全可変長マシンサイクル時間方式は種類数m の増加に伴って平均命令実行時間は短くなるが、量子化可変長マシンサイクル時間

方式では必ずしもこのようにはならない。さらに量子化可変長マシンサイクル時間方式の平均命令実行時間は完全可変長マシンサイクル時間方式よりも 1.6～7.0 %長くなった。実験機では完全可変長マシンサイクル時間方式を採用し、8 種類のマシンサイクル時間をもたせて製作したところ、 1%以下のハードウェアの増加で、平均命令実行時間は固定長マシンサイクル時間方式の59.6%となり、ギブソンミックスによる平均命令時間は2.5μsであった。

次に、設計時のマシンサイクル時間を、使用環境に適合するようにチューンアップすることによって、性能がどの程度改善されるか予測したところ、完全可変長マシンサイクル時間方式の性能改善率が量子化可変長マシンサイクル時間方式よりも大きいことが判った。言い換えると、これは、量子化可変長マシンサクル時間方式よりも完全可変長マシンサイクル時間方式の方がマイクロ命令実行分布の性能に対する影響を多くうけることを意味している。前述の設計したCPU のマイクロ命令実行時間分布が設計時と異なる数例に対して、チューンアップによる性能改善率をシミュレーションにより求めると最大

6%程度であった。さらに、このチューニングを自動的に行う回路構成を示した。

参考文献

1)S.S.Husson:Microprogramming:Principles and Practices,Prentice Hall(1970)

2)井上：ハードウェアの評価，情報処理，Vol.13,No.11 pp746-751(1972)

3)関野、徳永：システム評価技術，信学誌，Vol.62,No.11,pp1269-1274(1979)

4)溝口：マイクロプログラム・コントロール方式とその設計，情報処理，Vol.14,No.6,pp379-387(1973)

5)岩根、佐藤、溝口：マイクロプログラム制御計算機のタイミング設計における一考察，信学技法，Vol.79,No.2,EC79-83(1980)

6)岩根、佐藤、溝口：マイクロ命令の実行時間分布に基づく最適マシンサイクル時間の決定法，信学論，Vol.63-d,No.12(1980)

7)C.V.Ramamoorthy and H.F.Li:Pipeline Archtecture,Comput. Surveys,Vol.9,No.1,pp61-102(1977)

第５章　アベイラビリティ向上のための回路設計

５．１．はじめに

第２章の議論より、次の方策を採用すればRAS技術のためのハードウェアを少なくして信頼性を向上できることが判った。

マイクロプログラムによるオンライン検査において、マイクロプログラム検査プログラムのための制御記憶容量を削減し、かつ検査プログラムによるシステムリソースの消費を殆んど零にするために、CPUの遊休時に間接的に制御記憶のマイクロ命令を指定して実行させる。

システム立ち上げ、故障検出、故障回復及び故障修復に使用される保守用ハードウェア量を削減するために、これらに必要な機能を基本的な機能に分解してそれぞれを診断コマンドとしてまとめて、診断コマンドを解釈するハードウェアを用意する。

SVPの信頼性を向上させるために、低速小規模マイクロコンピュータを内蔵させ、メモリを48Kビットにし、また操作性を向上させるために、ソフトウェアを工夫する。

そこで、マイクロプログラム、診断コマンド及びSVPソフトウェアの機能分担を明確にし、上述の思想に基づいてCPU及びSVPを開発した。

本章では、まず開発したシステムの構成、中央処理装置について述べる。次に診断コマンドと診断コマンドを解釈実行する診断部及び保全機能を向上させるために特別に設けた診断パスについて述べる。さらにサービスプロセッサの構成と基本ソフトウェアについて述べ、最後にRAS用ハードウェア量について検討する。

５．２．システムの概要

５．２．１．システム構成[1]

開発したシステムは、中央処理装置（CPU）、システム制御装置（SCU）、主記憶装置（MMU）、チャネル制御装置（CCU）、サービスプロセッサ（SVP）及び各種周辺装置で構成され、図5.1にその構成図を示す。CPUは演算処理部（EPU）と診断部（DU)から構成され、EPUはマイクロプログラム制御パイプライン処理方式のプロセッサでMMUにあるプログラムを実行し、DUはEPUと独立に動作し後述の診断コマンドを解釈実行してEPUを制御する。SCUは機能分散された各種装置とMMU間及び各種装置間で競合が生じないようにデータやコマンドの授受を行い、CCUは各種入出力装置とMMU間でデータを多重化（Multiplexing)して転送する。SVPはシステム保全を行うサービスプロセッサであり、２台のCPUをサポートできる構造になっている[2]。

CCUとSVP間のインタフェイスをチャネルインタフェイス（CI)と呼び、オペレーティングシステム（OS)並びに其の他のプログラムと操作員とのマンマシンインタフェイスとして使用され、従来のシステム操作卓とCCU間の

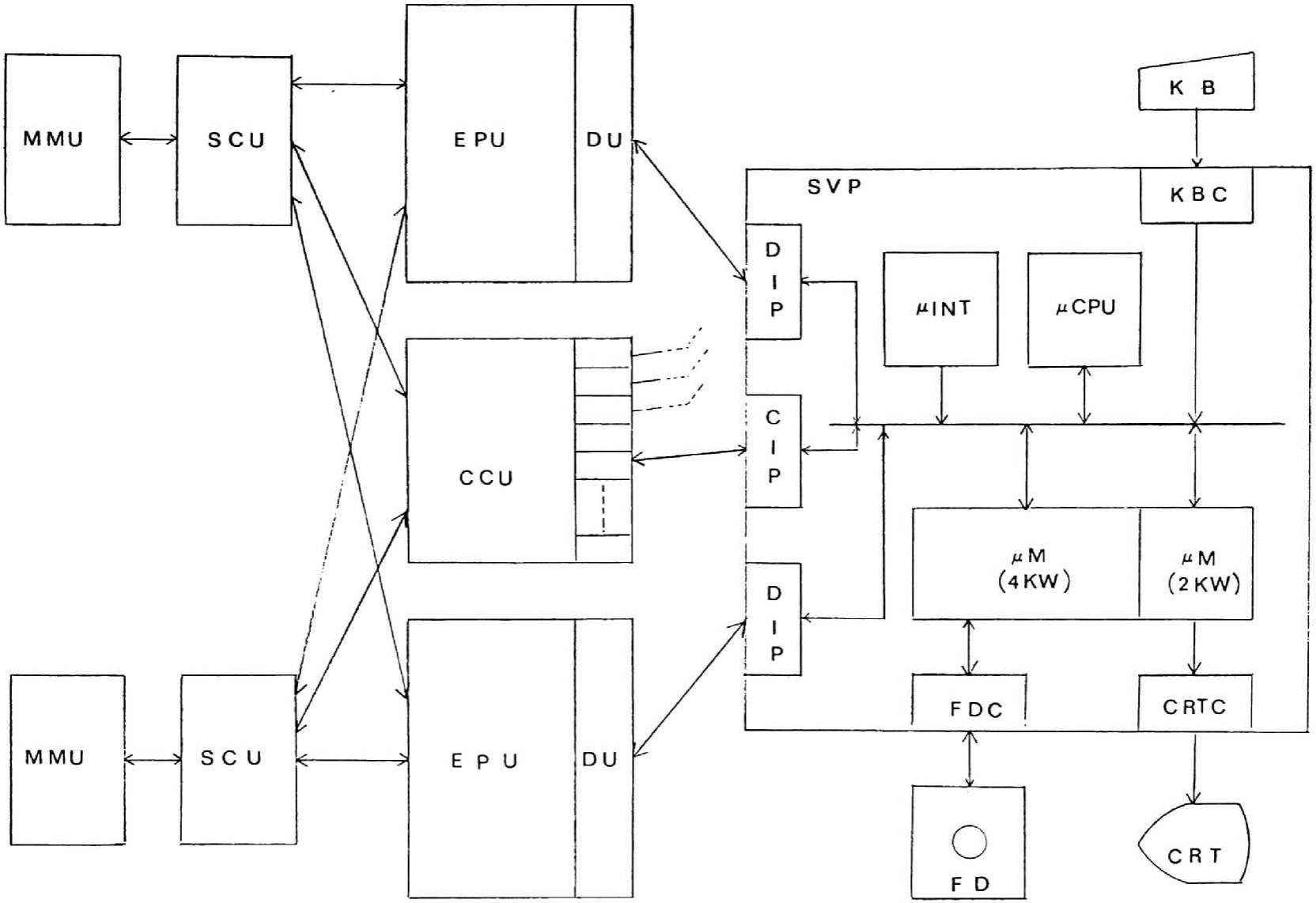

MMU
SCU
EPU
DU
CCU
MMU
SCU
EPU
DU
K B
SVP
KBC
DIP
CIP
DIP
μINT
μCPU
μM
(4KW)
μM
(2KW)
FDC
CRTC
FD
CRT

図5.1 システム構成

インタフェイスと同一である。また、CPUとSVP間のインタフェイスを診断インタフェイス(DI)と呼び、CPUの保守診断等に使用される。

5.2.2. 中央処理装置演算制御部(EPU)

本EPUは、仮想記憶を持っており、かつ命令語の先回り制御をパイプライン処理している。1命令語を実行するために必要な仕事は、命令語の読み出し、オペランドアドレスの計算、仮想アドレスから実アドレスへの変換、バッファ記憶からオペランドの読み出し、命令実行という5ステップに分解できる。これらのステップを各々Iサイクル、Aサイクル、Vサイクル、Cサイクル、Eサイクルと呼ぶ。命令語の読み出しは、命令語のアドレス計算、仮想アドレスから実アドレスへの変換、バッファ記憶から命令語の読み出しと細かく分解できるので、Iサイクルは A,V,Cサイクル用ハードウェアで処理できる。A,V,Cサイクルの処理は画一的なものが多いので、その大部分の制御を結線論理方式とし、Eサイクルの処理は多種多様な演算操作を行うので、マイクロプログラム制御方式とする。なお、バッファ記憶、高速アドレス

変換記憶及び主データパスはパリティ検査によりデータ保全を行う。また EPUハードウェア構成をマイクロ診断[3]のスタートスモール手順に適した構造となるように設計する[5]。

マイクロプログラムが格納される制御記憶（CS）は書換え可能であり、1語は80ビットで構成[4]され、8K語の容量を持っている。また、1語は A,B,C,Dの 4ブロックに分割されており、詳細を次に示す。A0-A3(Branch Control)ブロックはマイクロプログラムの順序制御及び定数処理を行い、A4-A7(Control Unit Control) ブロックはパイプライン制御及びメモリコマンド制御を行い、C0-C7(Execution Unit Control) ブロックは各種演算処理を行い、D(EDAC) ブロックは誤り訂正を行うためのものである。B0-B3(Execution Unit Small Control) にはC0-C7ブロック内の使用頻度の高いマイクロ操作を置き、B4-B7(Control Unit Small Control) ブロックにはA0-A7 ブロック内の使用頻度の高いマイクロ操作を置くことによってマイクロプログラムの削減を図っている。図5.2 にCPU の構成を示し、図5.3 にマイクロ命令語の構成を示す。

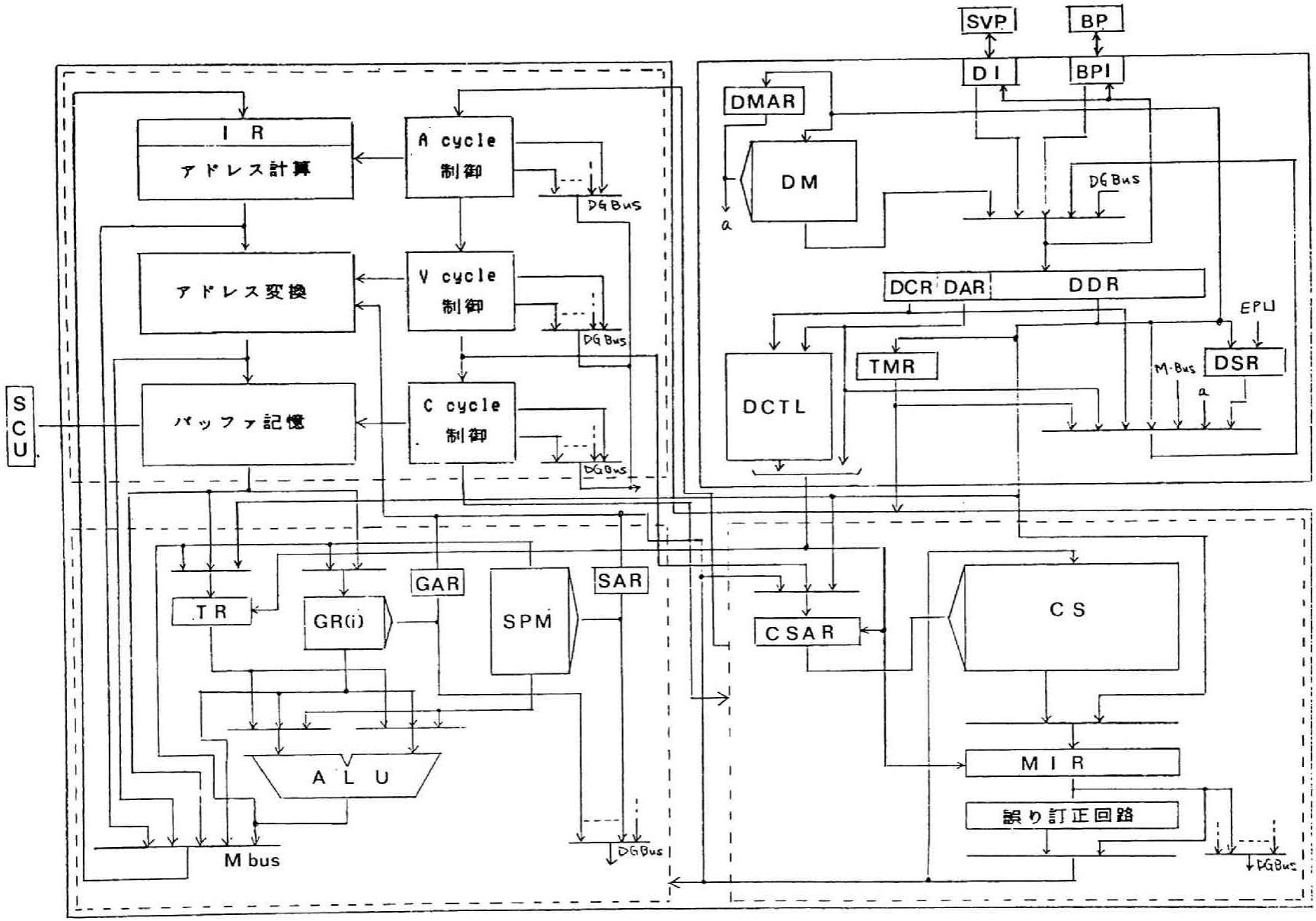


図5.2 CPU ブロック図

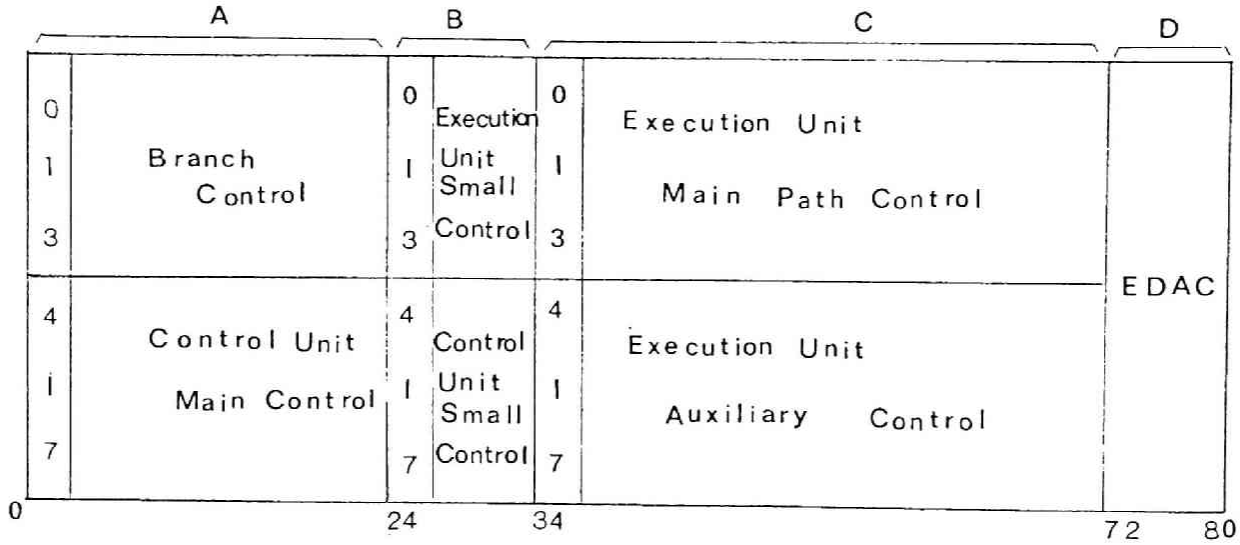


図5.3 マイクロ命令フォーマット

５．３．保全用回路の共通化

５．３．１．診断コマンド

故障検出、再試行、故障診断、故障修理、再立ち上げという故障回復及び修復処理は、レジスタの内容を読み出すこと、レジスタを任意の状態に設定すること、クロックを制御すること、EPU を初期化すること等の基本動作の組合わせにより実行できる。例えば、図5.2 においてGR(0) からALU を介してTRにデータが正しく転送されることを検査するために、次の手順が必要となる。但し、GR(0) に任意のデータが正しくできることは既に検査されているものとする。まず、EPU を初期化してEサイクル動作を実行可能とすると共に既知状態にする。GR(0) に検査データを設定する。次に、「GR(0) をTRに転送」するマイクロ命令をマイクロ命令レジスタMIR にセットした後に１ステップEPU を動作させることによって、GR(0) の内容がTRに転送される。そしてTRを読み出し正解値と比較することによって検証できる。ここで、GR(0) へのデータの設定及びTRの内容の読み出しは、EPUの命令実行のために用意されているハードウェアを利用できる。

このように故障回復及び修復処理を、書き込み、読み出し、クロック制御、初期化の基本的な動作に分解できるので、この動作の各々を一つの診断コマンドとしてフォーマットを統一してまとめることにより、診断コマンドの授受、解釈、実行等の制御を簡単にすると共に故障回復及び修復処理における各種制御手順の設計を容易にできる。

表5.1に診断コマンドのフォーマットと種類を示す。表5.1のSBPはStart Bit Positionで診断コマンドのデータの長さを表し、SBPによって不必要なデータの転送や記憶容量を削減できる。表5.1の診断コマンドの主要なものを次に説明する。

Start EPUコマンドはDU内モードレジスタTMRによって指定された初期化条件、実行開始条件でEPUのクロックを起動した後、TMRで指定された条件でEPUを停止させる。Step EPUコマンドは Start EPUコマンドと殆ど同じであるが、EPUのクッロクを1ステップのみ動作させる点が異なる。Read CSコマンドも Start EPUコマンドと殆ど同じであるが、EPUのクロックを動作させない点が異なる。特にこのコマンドは EPUの初期化にも使用で

表5.1 診断コマンド一覧

コマンド種類	サブコマンド	フォーマット					
		command	address	data 0	data 1	data 2	data 3
		0-2345670	0 — 34 — 7	0 — 7.0	7.0	7.0	7
Start EPU	———	0	制御記憶番地	———			
Step EPU	———	1	制御記憶番地	———			
Test EPU	Stop Test	20	診断記憶番地	———			
	Branch in Test	22	診断記憶番地	———			
	Start Test	23	診断記憶番地	———			
Read CS	———	3	制御記憶番地	———			
Read Data	Bus/Register	40	SBP レジスタ番地	SBPに従う			
	Register File	42	SBP ファイル番地	ファイル内番地	SBPに従う		
Write Data	Data Field	50	SBP レジスタ番地	SBPに従う			
	Data Register	52	SBP レジスタ番地	(DOR)			
Read Main Memory	———	6	SBP	———	SBPに従う		
Write Main Memory	———	7	SBP	———	SBPに従う		

(8進表示)

きる。

Start Testコマンドは DU内診断記憶 DMに格納されている診断コマンド列の実行を指示する。DM内のこのコマンドは無条件分岐を意味する。Stop Test コマンドは DM内の診断コマンドの実行を停止する。Branch in Testコマンドは Zeroインジケータを調べて零でないならば指示された DMの番地へジャンプする条件付飛越し命令である。これらのコマンドは後述の PATROLで主に使用する。

Read Bus/Regコマンドはアドレス部で指定したレジスタ及びバスの内容を DUに読み込む。EPU の多くのレジスタの内容は適当なマイクロ命令によってメインバス Mbusに出力できるので、後述の Write Dataコマンドによって目的レジスタから Mbusまで出力するマイクロ命令を MIRに書き込み、次に Read Bus/Regコマンドで Mbusを読み込むことによって、目的レジスタの内容を DUに読み込むことができる。また、EPU 内制御用レジスタの多くはマイクロ命令によって Mbusに出力できないので、後述の診断バス DGbus を使用して DUに読み込む。Read Reg File コマンドはレジスタファイル内アドレスを指定できることを除いて Read Bus/Reg コマンドと同じである。Read D

ata コマンドで直接読み込むことができるレジスタ及びバスはDU内レジスタ、Mbus、DGbus である。

Write Data Fieldコマンドは診断コマンドのデータ部をアドレス部で指定されたレジスタに書き込む。Write Data Regコマンドは診断コマンドデータレジスタDDR の内容を指定されたレジスタに書き込む。特に、Write Data Regコマンドは EPU内レジスタの内容をDUを介してEPU の他のレジスタに転送することができる。Write Data コマンドで直接書き込めるレジスタは、DU内レジスタ,MIR,CSAR,CS,TR,Address Trap Reg,Fault Trap Reg,Fcon Regである。Address Trap Reg及びFault Trap Regは、ソフトウェアのデバッグ及び保守修復のために用意されており、Fcon Regは再構成のために用意されている。なおアドレス部でCSが指定された時はデータ部は無視されてMIR の内容がCSに書き込まれる。

EPU 内のレジスタへのデータの書き込みは、Write Datacコマンドで一旦TRレジスタに書き込み、このレジスタから目的レジスタまでのゲートを開くマイクロ命令をMIR にWrite Dataコマンドで書き込み、次にStep EPUコマンドでEPU を1ステップ動作させることにより行う。

本方式の開発思想は、既存のハードウェアをできるかぎり利用して故障検出、故障回復及び修復用ハードウェアを極力少なくすることであり、上記のMIRに書き込まれるマイクロ操作はEPUの命令実行のために用意されたものである。

５．３．２．中央処理装置診断部（DU）

DUは、診断制御回路DCTL、診断コマンドコードレジスタDCR，診断アドレスレジスタDAR，診断データレジスタDDR，2Kバイトの診断記憶DM，診断記憶アドレスレジスタDMAR，モードレジスタTMR，診断状態レジスタDSR，DIインターフェイスのためのポートDIP，SVPが使用不可能のときマンマシンインターフェイスを司る最小の機能を持つ基本保守パネルBPと接続するためのBPIP及びEPUと情報の授受を行うEPUインターフェイスから構成される。

TMRはEPUの実行動作の制御を行い、実行開始条件、実行停止条件、初期化条件、その他検査回路の有効／無効等の設定を行う。実行開始条件には、MIRの内容をマイクロ命令として実行するもの、CSARで指定したCSの内容をMIRにロードして実行するもの、診断コマンドで指

定されたCSアドレスをCSARにセットした後CSARで示されたCSの内容をMIRにロードして実行するものがある。実行停止条件には、無停止、1命令実行停止、1マイクロステップ実行停止、アドレス一致停止、例外事態発生一致停止がある。初期化条件には、無初期化、制御回路のみの初期化、全回路の初期化がある。

故障修復処理及びOS開発において、EPUの内部状態を破壊せずにEPU内レジスタにアクセスする必要がある。EPU内レジスタの殆どは直接アクセスできないが、次の手順によりEPU内部を破壊せずに任意のレジスタにアクセスできることを示す。

EPU内レジスタの内容を読み出すためには、TMRを1マイクロステップ停止にしてEPUを停止させた後MIRに適当なマイクロ命令をセットして目的レジスタの内容をMbusに出力させ、次に診断コマンドでMbusをDUに読み込む。この操作でEPU内に状態変化のあったレジスタはMIRのみである。次に、今まで実行してきたEPUの状態から再開させるためには、CSARで指定されたCSのアドレスの内容をMIRにロードする開始条件をTMRに設定した後Start EPUコマンドを実行することにより可能である。

EPU 内レジスタへのデータ書き込みは、目的レジスタへの書き込みのためのクロックが必要となり、このクロックによって目的レジスタ以外のレジスタの状態まで変化する可能性がある。これに対して、TMR のパイプライン処理の禁止ビットをセットして、 Eサイクル終了後すなわち1命令実行停止後に、診断コマンドでTRにデータを書き込み、MIR にTRから目的レジスタまでのゲートを開き EPUを1マイクロステップ動作させることによって目的レジスタに書き込む。このとき EPU内の他のレジスタの状態が変化するが、次に実行するのは Aサイクル処理であるので制御回路の初期化を指定して EPUを起動させる。この方法では前の命令の Eサイクル終了後にしか目的レジスタに書き込めないが、実用上何ら問題は生じなかった。

DSR は診断状態レジスタで、EPU の実行中または停止、CS誤り検出、パリティ誤り検出、例外処理中に例外事態の発生(Fault on Fault)、PATROLで誤り検出等を表示する。

DUは EPUとは別系統のクロックで動作しているので、EPU が故障してもその影響を受けずにDUは動作できる。

DUの動作モードには、標準モードとDM制御モードとがある。標準モードはDIP またはBPIPからの診断コマンドを実行するモードで、DM制御モードはDM内の診断コマンドを実行するモードである。Start TESTコマンドの実行或はExecute PATROLマイクロ操作によってIDLEフリップフロップがセットしたときに、標準モードからDM制御モードへ移行し、DM内のStop TEST コマンドの実行によりIDLEフリップフロップがリセットされてDM制御モードから標準モードに戻る。DM制御モードでは SVPからの診断コマンドの授受を禁止するためにDIインタフェイスをBusyにしてある。なお、DM内の情報の保護のために 1バイト毎に 1パリティビットを持っている。

５．３．３．診断バス（DGbus）

診断バスDGbus は少ない入出力ピンで同時に多くの情報を読み取るために特別に保全用として設けた。DGbus はプリント基板PCB 毎にマルチプレクサ回路MPX を設けその出力をWired ORで接続することによって構成されている。Wired OR構成のために浮遊容量が大きくなって信号伝達速度は遅くなるが、DUは高速性を必要としないの

で実用上何ら問題はない。DGbusはRead Dataコマンドのアドレス部で指定でき、アドレス部の上位 nビットをPCBの選択信号、下位(8-n)ビットをMPXの選択信号として使用することにより、各PCBへの制御信号は(9-n)本となる。また、データ線を 8ビットにすることによってDGbusのために (17-n)本の信号線が各PCBへ接続されている。

DGbusは EPUの命令実行機能には使用されないので故障しても問題ないが、そのまま放置しておくと EPUが故障したときに使用できない事態が生じるので、システム立ち上げ時に SVPにより動作確認すれば解決できる。

５．４．サービスプロセッサの設計

５．４．１．低速小規模マイクロプロセッサの利用[6]

SVPは、フロッピーディスクFD，CRTディスプレイCRT，キーボードKB，CIポートCIP，2個のDIポートDIP及びこれらを制御するマイクロプロッセサuCPUと割込みラッチ回路uINT，メモリuM等から構成されたプログラム内蔵の独立した処理装置である。

uCPUは、12ビット並列プロセッサで、基本的に4bitsの命令コードを持ち、1.2MHZのクロックで動作し、レジスタタイプの通常命令を約7μsインダイレクトインデックスタイプの命令を約20μsで実行する。4k語のメモリ領域に対して直接アクセス可能であり、また 8レベルの割り込み受付とそれらの優先度を決定する機能を持っている。

プログラム用のメモリとして4k語を持ち、2k語はROMで構成され優先順位の高いプログラムを格納している。残り2K語はRAM で構成され、システムテーブルやオーバレイ領域等として使用される。これらのメモリにはパリティビットを付加している。メモリレイアウトを図5.4

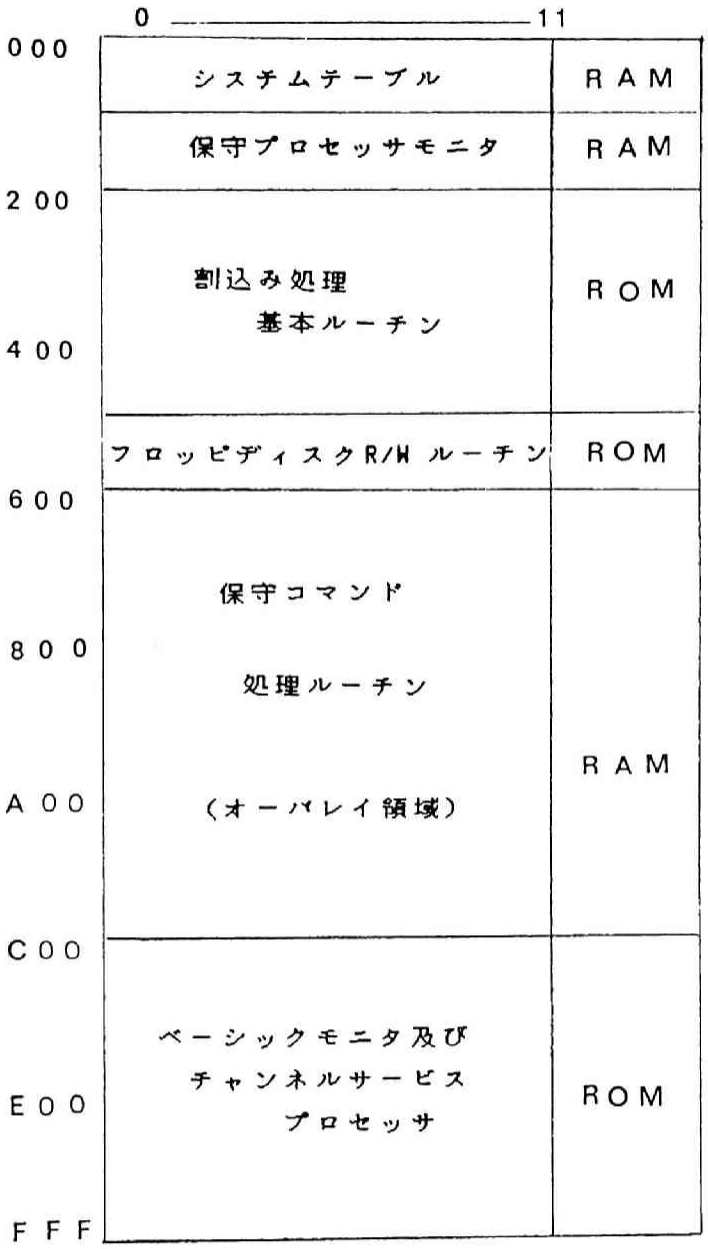


図5.4 SVP メモリレイアウト

に示す。

フロッピーディスクはメモリとDMA(Direct Memory Access)で接続され、巡回符号による誤り訂正を行っている。

５．４．２．ＳＶＰの基本ソフトウェア構成

SVPは、システム操作卓機能を実行するチャンネルサービスプロセッサ、保守修復機能を実行する保守プロセッサ及びシステム立ち上げやCPUの異常を処理し回復する割り込み処理プロセッサ、さらにSVP全体を制御するベーシックモニタからなり、図5.5にソフトウェア全体の構成を示す。

SVP処理モードには、単一処理モードと共用処理モードがあり、単一処理モードのときはチャンネルサービスプロセッサのみが動作し、共用処理モードのときはチャンネルサービスプロセッサと保守プロセッサが交互に動作する。チャンネルサービスプロセッサと保守プロセッサ間の制御の移行は一つのチャンネルメッセージまたは一つの保守コマンドの終了時点で起り得る。マルチCPUシステムにおいて、一つのCPUに故障が生じ、残ったCP

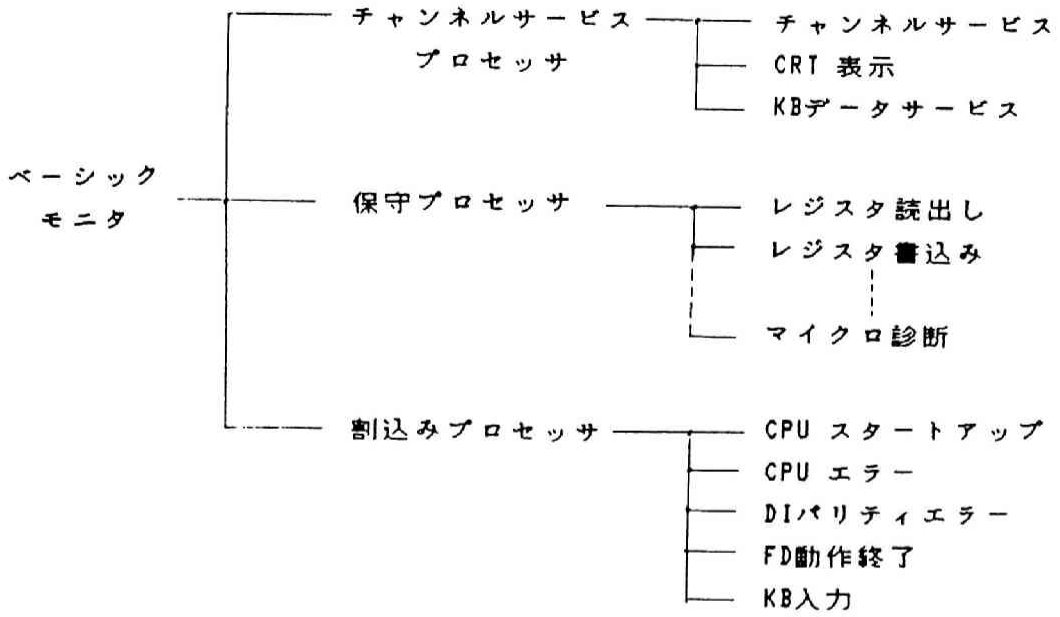


図5.5　SVP　ソフトウェア構成

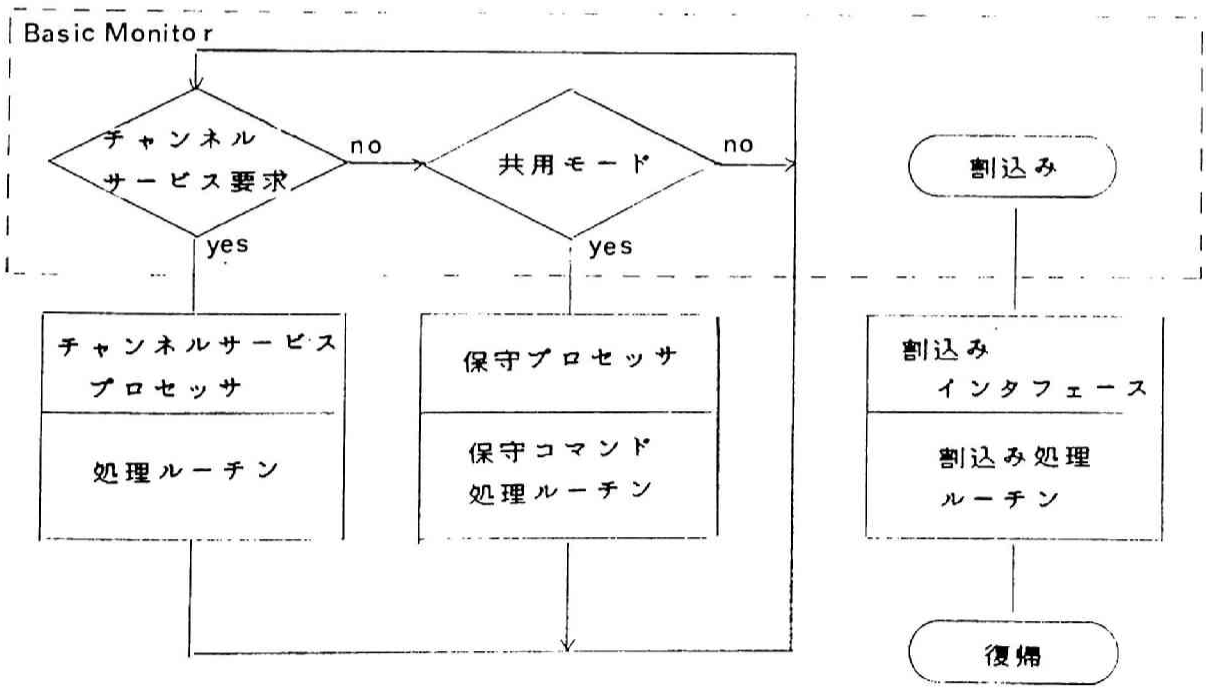


図5.6 ＳＶＰ　フローチャート

Uでシステムを再立ち上げしてジョブの継続する時、同時に故障したCPUを修復する必要がある。こういう場合に共用処理モードを使用すれば、1台のSVPですむのでコスト面で優れている。図5.6にSVPの基本動作のフローチャートを示す。

5.4.3.診断インタフェイス(DI)

DIインタフェイスは両方向に転送できる1バイトのデータ線とデータ転送を制御する3本の制御線を持ち、SVPが主導権をとって1バイトずつ転送するインタフェイスである。診断部DUがDIを介して受け取ったデータにパリティエラーを検出したときSVPへ報告するPER(PARITY ERROR)と、CPU内に何らかの異常が発生したことをSVPに報告するINTERRUPT(INTRPT)の2本の異常報告線を併設しており、合計13本の信号線で構成されている。図5.7にDIインタフェイスを示す。

データを1バイト読み取る診断コマンドをSVPからDUへ送ったときのタイミングチャートを図5.8に示す。図5.8において、Select信号が1になる毎にSVPの入出力機器であるDUが選択される。R/W信号が0であればSVP

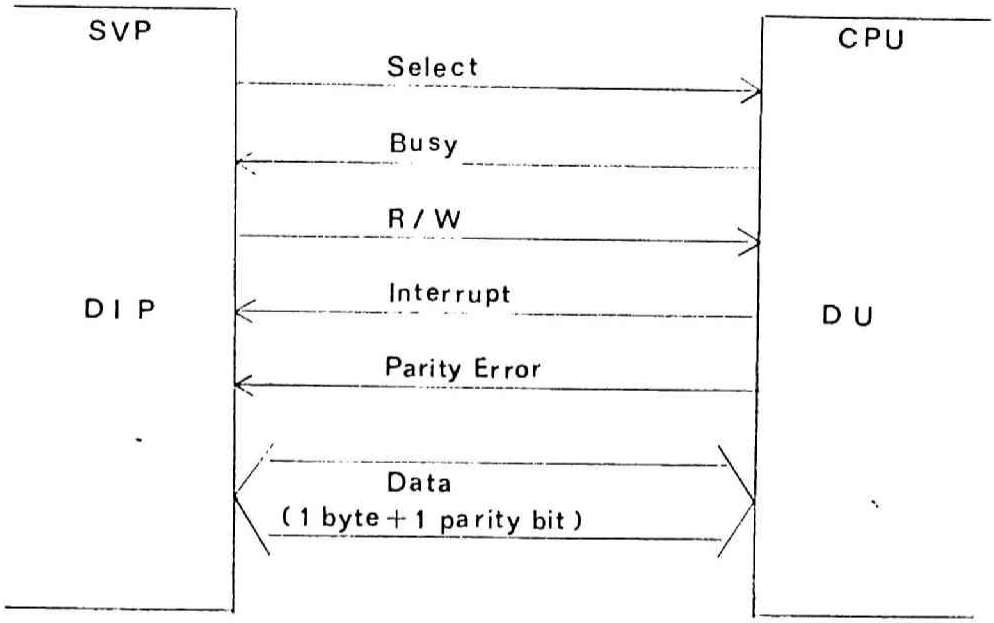


図5.7 DIインタフェースブロック図

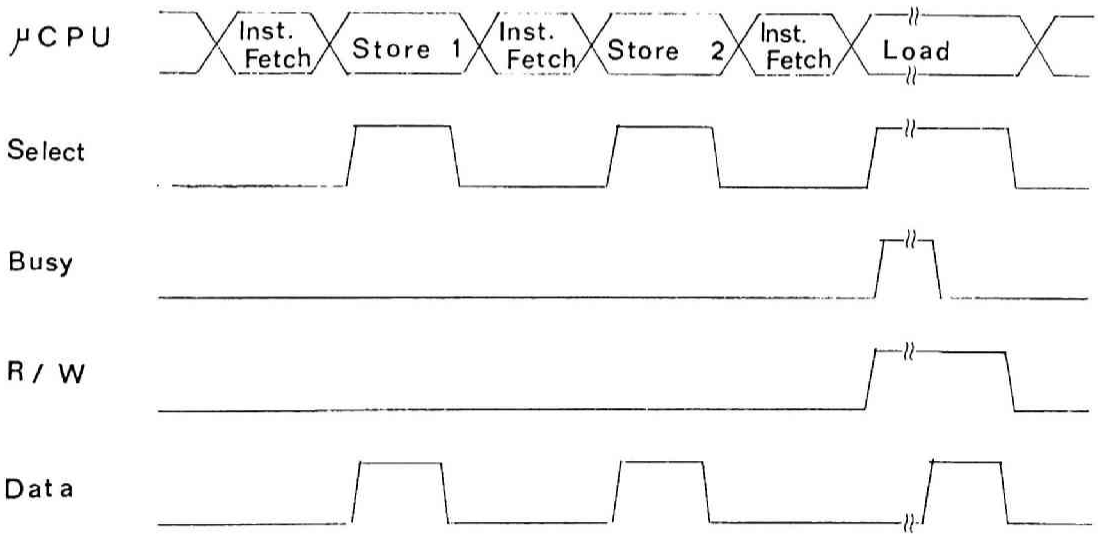


図5.8 DIインタフェースタイミング図

からDUへ、また 1であればDUからSVP へ 1バイトのデータが転送される。 SVPの指令に直ちに応答できないとき、DUはBusy信号を 1にする。SVP はBusy信号が 0になるまで待っており、 0になった時にDI上のデータを読み込む。図5.8 のIFはuCPUの命令フェッチサイクルを表し、Store1で診断コマンドコードをDIに送出し、Store2で診断コマンドアドレスをDIに送出し、そしてLoadでもってRead Data 診断コマンドを実行して得られたデータをDIからuCPUに読み込む。DUの実行速度は EPUと同じ高速論理素子を使用しているので、SVP と比べると数十倍速いので、SVP から 1バイト転送し、次の 1バイト転送するときにはDUの動作は通常完了している。そこでBusy信号のみでDIのタイミング制御が可能となり、一般のハンドシェイク形インタフェイスよりも簡潔になる。

５．５．開発結果並びに検討

前述の設計思想で開発したCPUは64Kビットのバッファ記憶、8K×80ビットの制御記憶及び16Kビットの診断記憶を除いて約12万ゲートで構成され、307種類の命令を実行し、マシンサイクル時間は294nsであり、その平均命令実行時間は0.6 MIPSであった。12万ゲートの内訳は、Aサイクル用ハードウェアが21000ゲート、Vサイクル用ハードウェアが26000ゲート、Cサイクル用ハードウェアが24000ゲート、Eサイクル用ハードウェアが40000ゲート及び診断部関連ハードウェア（制御記憶の誤り訂正等を含むRAS用ハードウェア）量はCPU全体の約7%であった。従来、高信頼化に費されるハードウェア費用は20～30%であると言われていた[7]が、本CPUでは約7%であり、かなり改善された。

また、制御記憶は、基本命令に1.7K語、事務用数値命令に1.7K語、事務用非数値命令に2.6K語、其の他の命令に1.2k語が実際に使用され、残りの0.8K語は機能拡張用として確保してある。このなかで、RAS機能のために専用に使用されるマイクロ操作はExecute PATROLのみである。

診断パスについては、約4000ゲートが搭載できる実装単位で 3本の選択信号が必要であった。

CPU内レジスタは約10個のレジスタと診断パスを経由して約64バイトの制御レジスタが診断コマンドのみで読み出せるが、マイクロプログラムと併用すると約 450個のレジスタと約64バイトの制御レジスタが読み出せた。また CPU内レジスタの約10個のレジスタは診断コマンドで直接書き込めるが、マイクロプログラムと診断コマンドを併用すると約 160個のレジスタに書き込むことができた。

SVP のメモリ容量は4K語であるが、ROM のうちで実際に使用したメモリは、ベーシックモニタ及びチャンネルサービスプロセッサに約 400語、フロッピーディスク読み書きに約 250語、キーボード入力及び CRT表示に約 200語、及び第6章で述べる割り込み処理に約 750語であった。残りの約 400語は機能拡張（例えばリモート保守のための回線処理等）のために確保してある。 診断部は1.44usの基本サイクル時間でもって 8サイクルで 1診断コマンドの受取りから実行までを行うように設計したので、SVP のマイクロプロセッサの約10倍以上の速度を

持っている。この速度差を利用してDIインタフェイスを簡潔にできた。また、本SVPは従来の操作卓に約10%のハードウェアを追加するだけで実現できた。

５．６．おわりに

本章では、マイクロプログラムを利用した故障検出、故障回復及び故障修復技術に必要な基本機能と実現するためのハードウェア量について議論した。

この基本機能を診断コマンドとしてフォーマットを統一し、命令実行用に設計されたマイクロ操作を保守診断用に組み合わせたマイクロ命令と、この診断コマンドを適当に組み合わせることによって故障検出、故障回復及び故障修復に必要な機能を実現できることを示した。なお、診断コマンドの解釈実行を診断部でまとめて行うようにした。

通常のマイクロ操作の組み合わせでは読み出せない制御用フリップフロップ等には診断パスを設けて診断部に読み出せるようにした。

このように設計することによって、120000ゲートで構成される CPUでは、診断部関連ハードウェアが全体ハードウェアの約 7%であり、従来に比べて削減できた。

またマイクロプログラムと診断コマンドを併用することによって、診断コマンドのみの場合と比較すると、レジスタの読み出しでは、約10倍、書き込みでは約15倍の

情報量を得ることができた。

SVPにおいて、プログラム用メモリを48Kビット用意したが、約4.8Kビットが機能拡張用に確保できた。またCPUとマイクロプロセッサの速度差を利用して、DIインタフェイスをハンドシェイク形よりも簡潔にできた。このようにしてSVPのハードウェアを小規模でかつ簡潔にして信頼性の向上を図った。SVPの機能及び操作性はソフトウェアに依存するが、これについては第6章及び第7章で議論する。

参考文献

1)F.Sato,Y.Uchida:The ACOS Series 77 TOSBAC System 600 Computer,Toshiba Review,No.114,pp23-27(1978)

2)岩根，内田，佐藤：ACOS 77/600 信頼性設計，第 8回日科技連信頼性保全性シンポジウム予稿集，pp233-236(1978)

3)岩根，佐藤，鈴木，高橋：ACOS 77/600 CPU のマイクロ診断方式，信学技法，Vol.78,No.26,pp25-34(1978)

4)岩根，佐藤：大型計算機における保全性設計，信学論 Vol.63-D,No.3,pp1050-1057(1980)

5)岩根，佐藤：故障診断に適したディジタルシステムの一構成法，信学技法，Vol.83,No.55,pp1-10(1983)

6)岩根，飯田，西中：オペレーションプロセッサ，昭和52年信学全大，1356(1977)

7)井原：システムの高信頼化技術，情報処理，Vol.23,No.4,pp327-334(1982)

第６章　共通保全回路を利用した故障検出及び回復技術

６．１．はじめに

間欠故障は、故障が固定故障に到る過程で起り、かつ固定故障より20倍以上多く発生すると言われている[1]。この間欠故障に対して誤り訂正及び再実行等の故障を隠蔽する故障回復が実用化されている[2]。この間欠故障が固定故障に変化してシステムダウンを引き起す前に、予防保守で修復しておくことが望ましい。間欠故障を早期に検出し局所化するために、マイクロプログラムによる検査プログラムを周期的かつ頻繁に実行する方法[3]があるが、この方法では、ハードウェア量の増加及びCPU性能の実質的な低下をもたらした。

第2章で議論したように、CPUの遊休時間に、間接的に命令実行用マイクロプログラムの一部のマイクロ命令を指定して実行させる方法を採用すると上述の欠点は改善される。

またCPUに故障による異常状態が発生したとき、CPU自身で故障回復を行うと成功しない場合が少なくない。この故障回復をSVPとCPUとが機能分担することによっ

てシステムダウンを減少できるが、この能力はSVPのソフトウェアに依存する。

本章では、まずマイクロプログラムによる自己検査プログラムであるPATROL(Processor Automatic Test Routine On Line)の基本原理及び基本動作について述べる。次にCPUの異常状態に対するSVPの異常処理及びSVPの割り込み処理について述べる。最後に、PATROL及び上述のSVPソフトウェアを開発して、それらをユーザーで使用した結果について検討する。

６．２．自己検査機構（PATROL）

６．２．１．基本原理[4]

入出力装置からの割り込み待ちでかつ他に実行するジョブがないとき、すなわち遊休状態のとき、CPUはDIS(Delay until Interrupt Signal)命令または自分自身への分岐命令を実行しており、単に外部割り込み(XIP)を待っている。このDIS命令を実行している間に検査プログラムを実行させることによって早期に故障を検出できる。

一般に、水平形マイクロプログラムの1マイクロ命令は64ビット以上で構成されている[5]。故障検出用マイクロプログラムを制御記憶CSに組込んだときはその容量が増加することと、任意の検査データをマイクロプログラムで作り出すのは余り容易ではないことが欠点である。そこで、あらかじめ診断記憶DMに診断コマンド列を格納しておき、そのコマンド列を実行することによて間接的に命令実行用の機能マイクロプログラムの一部を実行する手段をとると、2バイトで1マイクロ命令または1マイクロルーチンを実行させることができる。この関係を図6.1に示す。命令実行用マイクロプログラムの一部を適

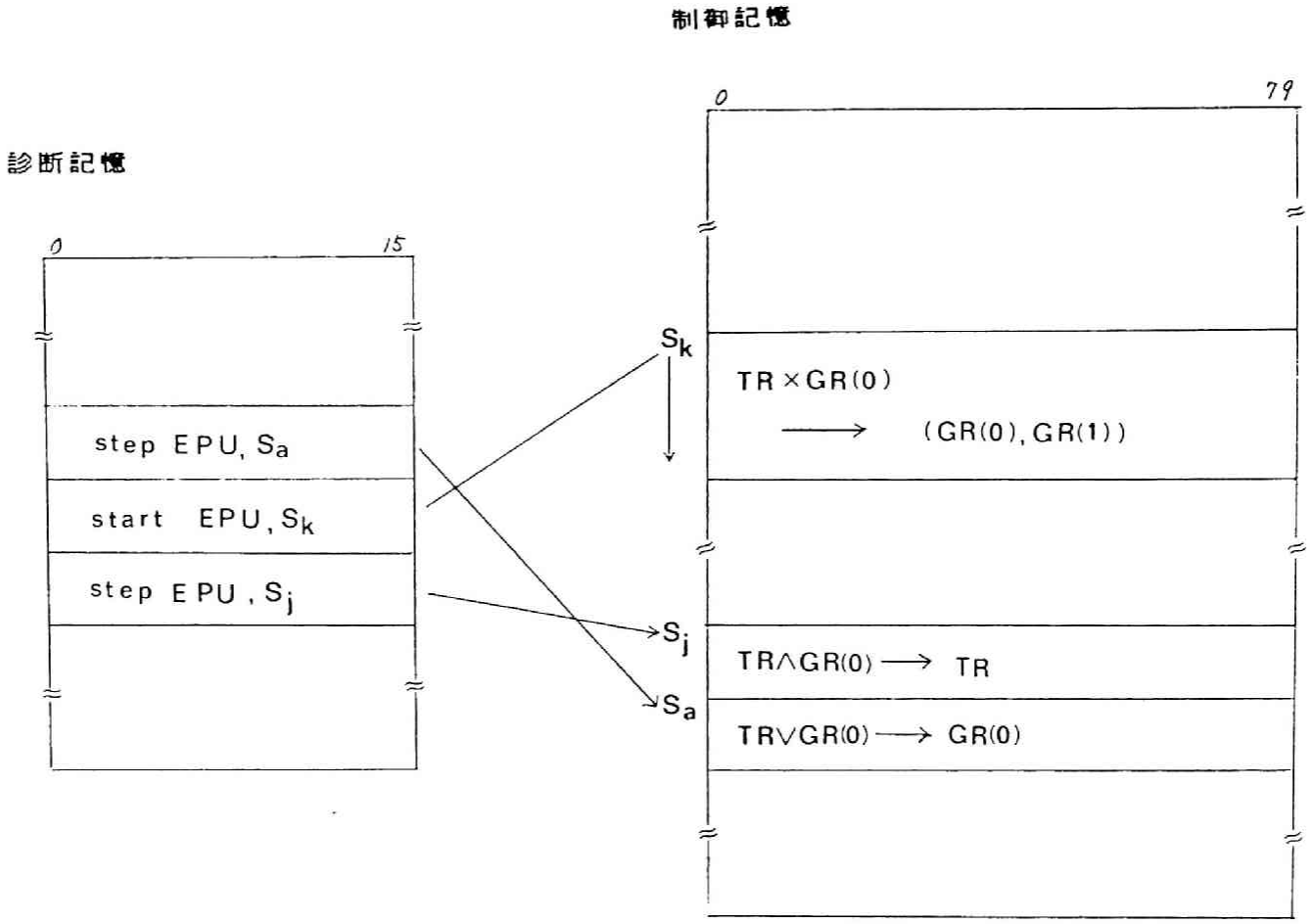


図6.1 診断記憶と制御記憶の関係

当に組合わせてつなぐことによって検査用マイクロプログラムが作成できる。この方法では診断コマンドがそのまま使用できるので、Write Dataコマンドで EPU内TRレジスタに任意のデータを設定でき、Branch in TESTコマンドで故障の有無を判定できる。そこで検査をTRから始めて検査結果がZero Indicatorに残るように検査マイクロプログラムを設計すればよい。なお、この方法では DIS命令の中でPATROLを開始させるためのマイクロ操作Execute PATROLが唯一つ必要である。またDMは書き込み可能な記憶であるので、任意の診断コマンド列を格納して実行することができる。

６．２．２．基本動作

図6.2にPATROLの基本動作を示す。図6.2の左のフローチャートは DIS命令のマイクロプログラムを表示し、右側の箱は診断記憶DMの内容を表している。

EPUが DIS命令を実行すると、まず DISフリップフロップをセットし外部割り込みXIP を調べる。 XIPが存在すれば DISフリップフロップをリセットして割り込み処理のマイクロルーチンを実行し、存在しなければExecut

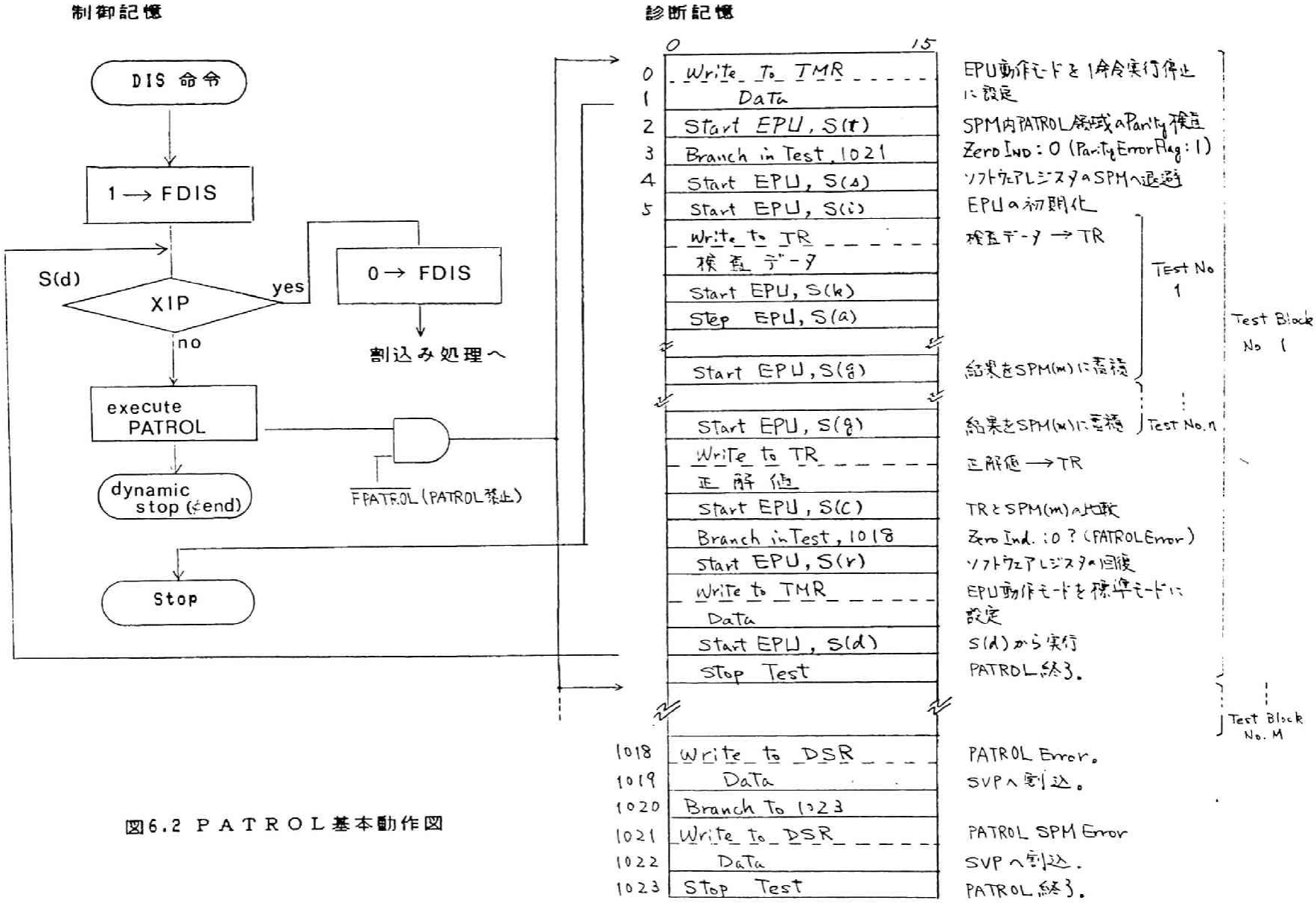


図6.2 ＰＡＴＲＯＬ基本動作図

e PATROLマイクロ操作を実行してIDLEフリップフロップをセットする。このとき、DU内TMR レジスタのPATROL禁止ビットがセットしていなければDUはDM制御モードになり、診断記憶アドレスレジスタDMARの指しているアドレスに格納されている診断コマンドから実行していく。一方 EPUは次のマイクロ命令(No Operation)を実行して動的停止の状態にある。PATROLに制御が移って最初に実行する診断コマンドはWrite Data to TMR である。このコマンドにより TMRを書換て EPUを1命令実行停止モードにして EPUを停止させる。次にスクラッチパッドメモリSPM 内のPATROL用ソフトウェアビジブルレジスタの退避領域の検査を行う。故障を検出したときはDU内状態レジスタDSR のSPM 故障ビットをセットして SVPに割り込み信号を送り、次にStopTESTコマンドを実行しDM制御モードから標準モードに移行する。故障を検出しなかったときは、Start EPU コマンドを実行してソフトウェアビジブルレジスタをSPM に退避させた後、Start EPU コマンドを実行して EPU内レジスタを既知の状態にする。

次に検査データをTRレジスタに設定して TEST NO.1の実行を開始する。TEST NO.1 の終了時点で EPU内各種レ

ジスタに記憶された検査結果をSPM(m)に貯える。このように順次検査を行い TEST NO. n の終了後の最終検査結果はSPM(n)に格納される。次にTRレジスタに正解値をセットし、TRとSPM(n)とを比較して一致すればZero Indicatorをセットする。そして、Branch in TESTコマンドを実行させて Zero Indicator をチェックし、セットしていなければ、これは故障を検出した場合であるので、DSR レジスタのPATROL故障検出ビットをセットしてSVP に割り込み信号を送った後にDUを標準動作モードに戻す。故障を検出しなかったときは、Start EPU コマンドでSPM に退避させたソフトウェアビジブルレジスタを回復させた後、Write Dataコマンドで EPU を 1 命令停止モードからNON STOPモードにして、次にStart EPU コマンドで DIS命令の外部割り込みの有無を調べるマイクロ命令から実行させる。

図 6.3 にPATROLでエラーを検出しなかった場合のタイミングチャートと故障を検出した場合のタイミングチャートを示す。

６．２．３．割り込み応答時間

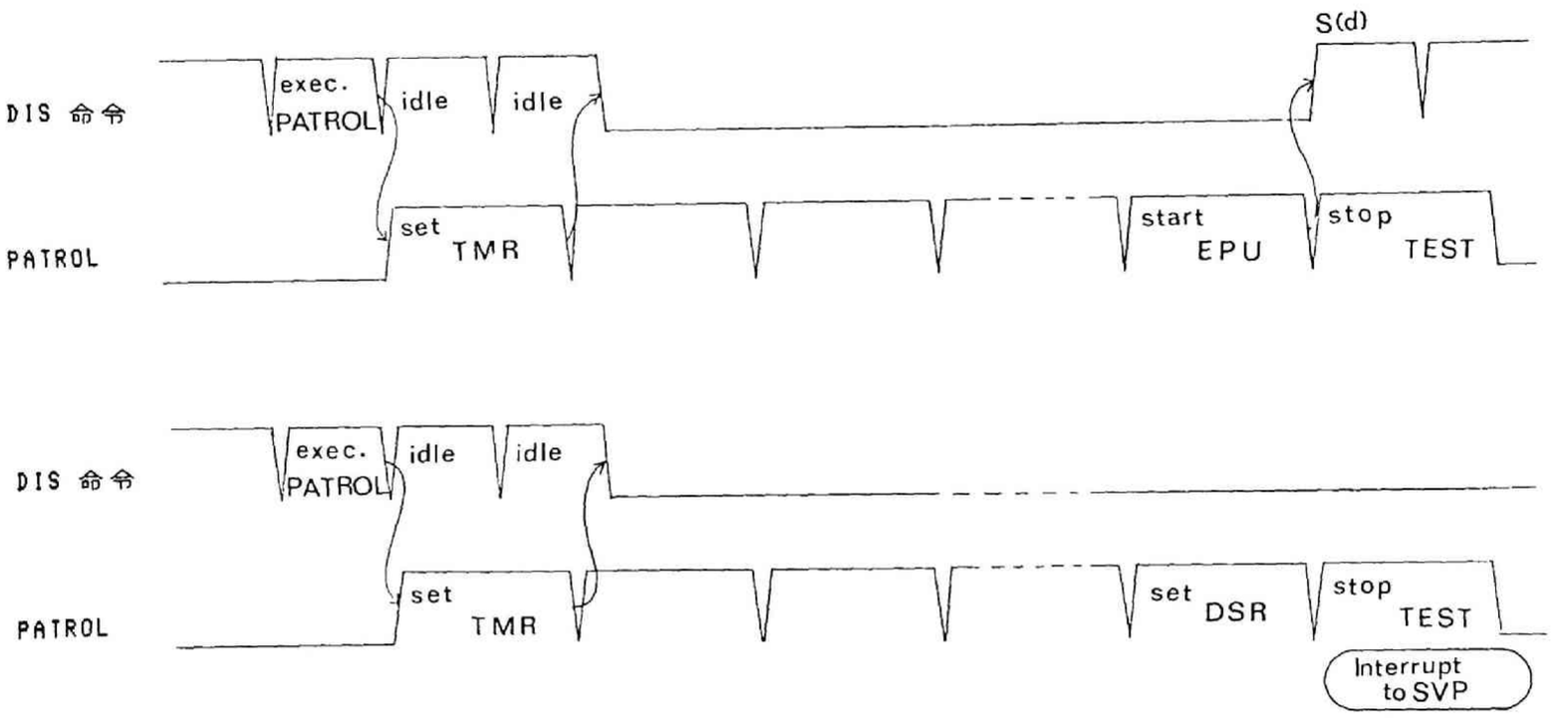


図6.3 PATROLとDIS 命令のタイミング図

基本動作では、DIS命令の最初のマイクロ命令で外部割り込み（XIP）の有無を調べた後で、PATROLを実行するようになっていた。しかし、PATROLの実行に入った直後にXIPがセットすれば、割り込み要求はPATROLの終了まで待たされ、結果としてシステムリソースを使用して検査プログラムを実行したことになる。そこで、検査プログラムを複数個の検査ブロックに分割して、各検査ブロックを一つの独立した実行単位にすることにより、各検査ブロック終了後に割り込みの存在を調べるマイクロ命令からEPUを実行するようにしておけば、割り込みの待ち時間は短縮される。図6.2では、検査プログラムがMブロックに分割された例を示している。

６．３．異常処理に対する設計

６．３．１．中央処理装置の異常状態

中央処理装置CPU に故障等による異常状態が発生したとき、CPU 自身で回復するように設計されていた[6]。この場合、故障した部分を使用しないで回復処理が可能ならばその回復処理は成功するが、故障個所を使用するならば成功する可能性は少ない。そこで、CPU の異常を検出したときに、サービスプロセッサ SVPに通知して回復処理の一部を SVPに任せることによってシステムダウンを少なくできる。

CPU に何らかの異常状態を検出したとき、DU内のDSR レジスタにその原因を表示するビットをセットして SVP に割り込み信号をおくる。DSR レジスタで CPUの異常状態の原因を表す主なものは、(1) 例外処理中に例外事態の発生，(2)制御記憶CSの2 ビット以上の誤り，(3)EPU パリティエラー，(4)PATROLによる故障検出，(5)PATROL実行時のソフトウェアビジブルレジスタ退避用SPM の故障，(6)診断記憶DMのパリティエラー等である。

割り込み原因の(1)，(2)，(3) は致命的な故障が発生していると予想されるので、マルチCPU 構成のシステムで

はこのCPU を切り離し再構成してシステムの稼動を継続し、シングルCPU では至急故障修復する必要がある。（4）の場合、EPU は遊休状態であるので直接被害を受けるジョブはなく、この故障が間欠的なものであれば故障修復は困難であるので、故障を局所化するPATROL検査プログラムを診断記憶にロードし、SPM に退避したソフトウェアビジブルレジスタを回復させ DIS命令の外部割り込み待ちマイクロ命令から実行させる。この故障が固定的なものであれば （1），（2），（3）と同じ処理を行う。（6） はCPUの本来の機能遂行には全く影響はなく、単に故障検出能力が低下するだけであるので、従来のソフトウェアによる検査プログラムを実行するように報告し、TMR レジスタのPATROL実行禁止ビットをセットして DIS命令の外部割り込み待ちマイクロ命令から実行させる。（5）の場合は検査結果の比較にEPU の演算回路を使用するのでソフトウェアビジブルレジスタの退避用SPM の故障とは断定しにくいが、この故障の可能性が大きいので（6）と同じ扱いとする。

これらを実行する手順は SVPのマイクロプロセッサの命令と診断コマンドによって容易に作成できる。

６．３．２．サービスプロセッサ（SVP）への割り込み[7]

SVP には、CPU#1 異常事態発生、CPU#2 異常事態発生、DI#1パリティエラー、DI#2パリティエラー、CPU 立ち上げボタンのプッシュ、フロッピーディスク動作終了及びキーボード入力等の 8レベルの割り込みがあり、その割り込みレベルに応じた処理ルーチンを持っている。特に CPU異常事態発生による割り込みは即座に処理される必要があるので、割り込み優先度の最上位におく。図6.4 にCPU 異常状態発生割り込み処理をフローチャートで示す。CPU 異常状態割り込み処理ルーチンが起動されると、最初にDIインタフェイスの動作確認が行われる。この動作に異常がなければ Read Data in DSR 診断コマンドによりDSR レジスタの内容がSVP に読み込まれる。次にDSR レジスタの各ビットを調べて、必要があればその割り込み原因に対応する処理ルーチンをフロッピーディスクからオーバレイ領域に読み込み実行する。終了後、未処理の割り込み原因があるかどうかを調べて無ければ割り込んだ場所に制御を戻す。なお、割り込み処理ルーチンのうちで短かい応答時間を必要とする部分を基本ル

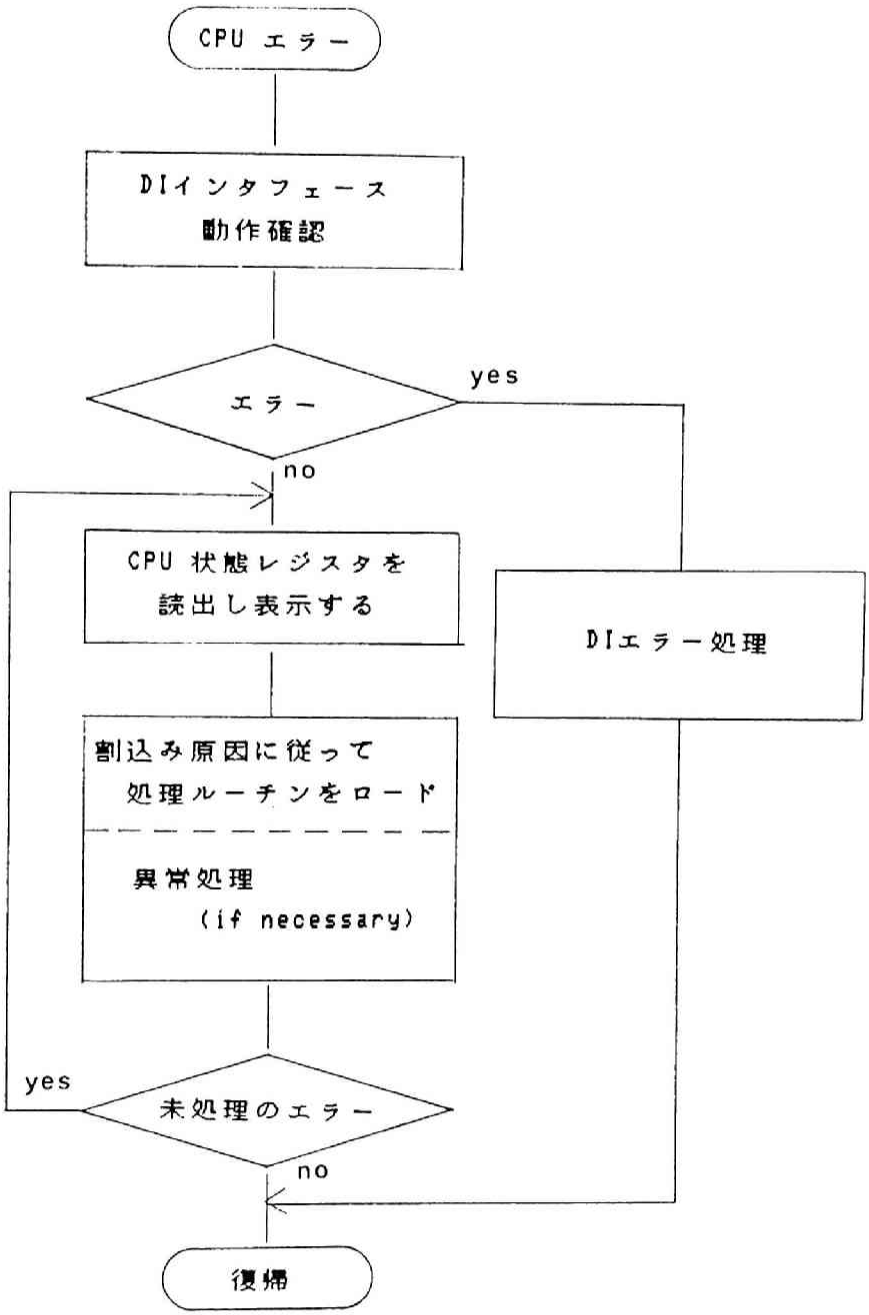


図6.4 CPU 異常処理フローチャート

ーチンとしてROMに格納した。

従来のOSだけでは困難とされていたCPU異常事態発生時の故障回復処理はSVPとCPU内診断部DUが適切に機能分担することによって、操作員の手間を煩わせることなく可能となった。また、PATROL検査プログラムを局所化できるように階層化しておいてPATROLで間欠故障が検出されたとき、より細かく故障個所を指摘する検査プログラムをPATROLとして実行させることによって間欠故障に対しても有効である。

６．４．開発結果並びに検討

幾つかのユーザーで実測した結果では、CPU が割り込み待ちの状態になるのは、平均して50msに 1回の割合でおこり、このうちの約 5% が DIS命令の実行状態にあった。すると DIS命令は 1秒に 1回実行されていることになる。

1診断コマンドの実行時間は11.52us であるので、1Kの診断コマンドの実行時間は11.52ms である。さて、DIS 命令のなかのXIP マイクロオーダを実行した直後に割り込み信号が入ったとする。すると、検査プログラムをブロックに分割しない場合では、この割り込みに対する処理は11.52ms 遅れることになり、これでは約1.15% システムリソースを使用したことになる。

そこで、検査プログラムを約10ブロックに分割し、各ブロック実行後は必ずDIS 命令に戻って割り込み信号をチェックするように設計した。その結果、最悪のときでも割り込み処理は1.15ms遅れるだけでシステムリソースの使用は無視できる程度となった。

ソフトウェアビジブルレジスタの SPMへの退避、 SPMからの回復及びこの SPMの検査ののために、命令実行に

用意されたマイクロ命令をつないで使用するのでは診断記憶を必要とするので、このために約60語のルーチンを作成して制御記憶に格納した。このルーチンは第 5章の制御記憶容量の評価では其の他の命令に含まれている。

フィールド技術者からの報告によると、PATROLで何等かの故障を検出したとき、数ヶ月後には殆んどの場合固定故障となって表面化した[8]。ただ、故障を局所化するPATROL検査プログラムが良くなかったために、間欠故障個所を特定できなかった。PATROLは検査プログラムをうまく階層化することによって間欠故障に対する有効な方式でもあると言える。

間欠故障の発生を表面化させることは、ユーザーに不安感を与えることになるので、再実行等で回復し、一方ではPATROLの故障局所化検査プログラムで記録をとりながら実行させるのが良いと考えられる。

標準のPATROL検査プログラムは約1000ステップで構成され、バッファ記憶及び高速アドレス変換記憶はパリティ検査を行いかつ故障を局所化しやすいので検査対象から除外し、残りの EPU内回路をPATROL検査対象とした。固定単一故障を仮定したときの故障検出率は約85%で、

CPU の性能低下は前述のように無視できる程度であった。また、診断記憶を採用することによって検査プログラムの格納容量を1/5 に削減できた。

SVP の割り込み処理用記憶として約 750語用意しておいたが、この領域には基本的な処理ルーチンを格納し、残りの部分をフロッピーディスクに置いた。割り込み処理で使用する全体の容量は約8Kバイトであった。

６．５．おわりに

本章では、命令実行用マイクロプログラムの一部を間接的に指定して検査プログラムを構成し、それを CPUの遊休時に実行させるPATROLによる故障検出技術、及び SVPを使用する故障回復技術について述べた。

PATROLの検査プログラムは、平均すると 1秒に 1回の割合で実行される。検査プログラムを約10個の検査ブロックに分割して実行させることにより、性能低下は約0.12%となった。またハードウェア量は従来のマイクロプログラムによる検査プログラム法の約1/5 となった。フィールド技術者からの報告によると間欠故障の早期検出に有効であった。

SVP の CPUの故障による異常処理を含む割り込み処理のプログラム量は全体で約8Kバイトとなった。割り込み処理プログラムを、短い応答時間を必要とする部分（基本ルーチン）とそうでない部分にわけて、基本ルーチンを約 750語の ROM領域に格納した。これにより故障回復処理も小規模マイクロプロセッサ内蔵の SVP の機能に組込むことができた。

参考文献

1)当麻：高信頼化技術，信学誌，Vol.62,No.11,pp1260-1269(1979)

2)久保，堀越：システム障害回復技法，情報処理，Vol.23,No.4,pp342-348(1982)

3)東芝：TOSBACシリーズ7/40システム概説書

4)岩根，佐藤：大型計算機における保全性設計，信学論 Vol.63-d,No.3,pp1050-1057(1980)

5)T.G.Rauscher and P.M.Adams:Microprogramming: A Tutorial and Survey of Recent Developments,IEEE Trans. on Comput.,Vol.C-29,No.1,pp2-20(1980)

6)小野塚，広江，阿部：「ACOS」シリーズ77「TOSBAC」システム600/700 信頼性システム，東芝レビュー，Vol.30,No.6,pp475-478(1975)

7)岩根，飯田，西中：オペレーションプロセッサ，昭和52年信学全大，1356(1977)

8)西中：TOSBAC月間フィールド信頼性情報ACOS-600S,東芝エンジニアリング社内報告

第７章　共通保全回路を利用した故障修復技術

７．１．はじめに

第２章及び第５章で議論したように、低速小規模マイクロプロセッサを内蔵することによってSVPの信頼性は向上する。一方、SVPのプログラムのためのメモリ容量が少ないが、フロッピーディスクのような補助記憶を使用すれば機能は実現できる。大規模なマイクロプロセッサを内蔵したSVPではレジスタをニーモニックで指示していた[1]が、小規模なマイクロプロセッサを内蔵したSVPでは番号で指示していた[2]ので操作性が良くなかった。またSVPソフトウェアとCPUの修復に必要な手順が分離されておらずCPUハードウェア変更に伴ってSVPソフトウェアを変更する必要があってソフトウェアの保守性が良くなかった。このような問題点を解決し、さらに小容量のメモリで、その操作性及び応答性において実用に耐えるように設計するためには工夫が必要である。　本章では、まず保守プロセッサについて述べる。マンマシンインタフェイスを良くするための操作員に分かり易い保守コマンド及びコマンド形式について述べ、この保守コ

マンドの実行時間について議論する。そして、簡単な低級言語を使用することによって、保守コマンド実行時間が実用的な範囲に収まり、CPUのレジスタの読み出し及び書き込み手順が分かり易く、かつCPUハードウェアの変更に容易に対処できる方法について提案する。次に、マイクロ診断プログラムの設計思想及び設計方法について述べる。保守プロセッサと同様に、マイクロ診断プログラムの実行時間を予測する。そして、簡単な低級言語を使用することによって、実用的な範囲にマイクロ診断プログラムの実行時間とプログラム容量が収まり、CPUハードウェアの変更に容易に対処でき、かつ検査手順がコンパクトになるような方法について提案し、最後に、これらの提案した方法により保守プロセッサ及びマイクロ診断プログラムを開発した結果について検討する。

７．２．保守プロセッサの設計[3]

７．２．１．保守コマンド

第 5章の図5.2 において、GRレジスタのGAR で示しているアドレスにデータを書き込むためには、次の手順が必要である。まず、 EPUを 1命令実行停止モードにし EPUを停止させる。 Write Data 診断コマンドでデータをTRレジスタにセットし、次に「TRレジスタの内容をGRレジスタに転送する」マイクロ命令をMIR にセットする。そして EPUを 1ステップ動作させる。このような手順を必要とする方法は人間にとって理解しにくい。そこで、故障修復処理に必要な機能を抽出し、人間にとって理解しやすい保守コマンドとしてまとめた。保守コマンドには、表6.1 に示すように EPUの起動、 EPUの初期化、構成変更、マイクロ診断等がある。特にMacro 保守コマンドは、幾つかの保守コマンド列を定義して登録したり、登録した保守コマンド列を実行する機能である。この保守コマンドはマイクロプログラムのパッチにも使用できる。

サービスプロセッサSVP はキーボードKBの「保守」キーが押されると共用モードになって画面上に 〈M〉を表示

表7.1　保守コマンドリスト

保守コマンド	説　　明
Macro	マクロコマンドの定義及び実行
Read	レジスタ又はメモリ内容の読み出し
Write	レジスタ又はメモリへデータの書き込み
Execute	EPU の起動
Step	1マイクロ命令又は 1命令実行モードの設定
Config.	システム構成の変更
Initialize	EPU の初期化
Adr Stop	制御記憶又は主記憶アドレス一致停止モード設定
Flt Stop	例外事態発生停止モード設定
Load	制御記憶又は診断記憶へローディング
Transfer	診断コマンド(Immediate form)の転送
Diagnose	マイクロ診断プログラムの実行
Bye	保守プロセッサ動作の終了

して保守コマンド待ちとなる。保守コマンドが入力されると、保守プロセッサはフロッピーディスクからオーバーレイ領域に該当する保守コマンド処理ルーチンを読み込み、保守コマンド処理ルーチンに制御を渡す。保守コマンド処理ルーチンは診断コマンド列からなる手順をフロッピーディスクから読み込んで診断部DUに送り、結果を画面上に表示する。

この時の保守コマンド実行時間Tは次式で近似することができる。

$$T \sim nt(a)+t(c)+t(d) \qquad [7.1]$$

ここで、t(a)は保守コマンド処理ルーチンまたは診断コマンド列よりなる手順をフロッピーディスクから読み込む平均時間、nはフロッピーディスクから読み込む回数、t(c)はマイクロプロセッサが診断コマンド処理ルーチンを実行している時間、t(d)は診断部DUが診断コマンドを実行する時間である。一般にt(a)<<t(c)<<t(d)であるので[7.1]式は次式となる。

$$T \sim (n+1)t(a) \qquad [7.2]$$

また、フロッピーディスクのヘッドの移動時間をt(m)、ヘッドロードの時間をt(l)、回転待ち時間をt(r)とす

ると、フロッピーディスクからの平均読み込み時間Tは次式で近似できる。

$$T \sim (n+1)(t(m)+t(l)+t(r)) \qquad [7.3]$$

マンマシンインタフェイスを考えると、保守コマンドを入力してその応答が画面上に表示されるまでの時間は短い方がよいが、実用面からは1.0秒弱程度の待ち時間は許容限度である。平均して10トラック移動すると、これに要する時間が約100ms、平均回転待ち時間が約80ms及びヘッドロード時間が約20msであるので、応答時間が1.0秒以内となるように計算すると、nは4以内である。

7.2.2. 保守コマンド処理ルーチンの設計

EPU内レジスタの書き込み及び読み出しのための診断コマンドによる手順は目的レジスタ毎に異なるので、指定されたレジスタに対する診断コマンドの手順のみをフロッピーディスクから読み込むようにしないとメモリに入りきらない。

Mbusを介して読み出された目的レジスタの内容をそのままの形で画面上に表示したり、目的レジスタへ書き込

むための経路を考慮してキーボードから書き込みデータを入力することはそれは人間にとって非常に分かりにくい。例えばマイクロ命令でARレジスタの内容を図7. に示した形でMbusに出力でき、このARレジスタの内容を画面上に表示する例を考える。この場合、次の手順が必要である。まず、Write Data診断コマンドで MIRに上記マイクロ命令を書き込み、Read Data 診断コマンドでMbusを 4バイト読み出す。次にARレジスタ部分を抽出するために 4ビット左シフトし、それをFF000000(16)でマスクする。そして、マスクしたものを左から 2バイト表示する。このような編集を行うことによってマンマシンインタフェイスが改善される。この編集はレジスタ毎に固有であり、個々のレジスタ毎の編集プログラムを組込むことは、プログラムが複雑になる上に CPUのハードウェアの変更があるたびに編集プログラムも変更しなければならない。そこで、レジスタ毎に異なる編集作業を基本的な幾つかの作業に分解し、制御語でその作業を指示することによって編集プログラムを簡潔にでき、更に診断コマンド列と保守コマンド処理ルーチンが分離でき CPUのハードウェアの変更にも診断コマンド列の変更だけで対

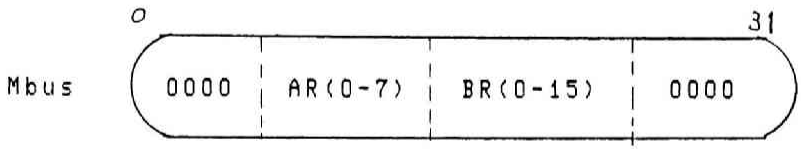
処できる。

表7.2に保守コマンド処理ルーチン用制御語を示す。ARレジスタをMbusを介して読み込み、画面上に表示するための制御語を含んだ診断コマンド列を図7.1に示し、制御語の役割を次に説明する。Name制御語は手順の先頭におかれ、他の手順との区別のために使用される。Nameに続くARはレジスタ名である。Read制御語は直後に続くWrite Data(AR → Mbus) to MIR及びRead Mbusが CPUからデータを受けとる診断コマンド列であることを表し、Rcv04制御語は 4バイトのデータを CPUから受けとることを示す。Shft1制御語は受けとった 4バイトのデータを 4ビット左シフトすることを指示し、Mask制御語は直後に続くFF000000でこのデータをマスクすることを示す。次のOut02制御語はマスクしたデータを左から 2桁をCRTに表示することを指示し、End制御語は手順の終を示す。

このように、制御語を含んだ診断コマンド列は簡単な低級言語で書かれており、診断アセンブラでビットパターンに変換されて、あらかじめフロッピーディスクに格納されている。

表7.2　保守プロセッサ用制御語

制御語	コード(16 進)	説　　明
Ident.(Name)	800	レジスタ、バス名を示し手順の先頭に置く。
Read(Read)	801	レジスタ、バス内容を読み出す診断コマンド列を示す
Write(Wrt)	802	レジスタ、バスに書き込む診断コマンド列を示し、DpntXXと共に使用する。
Data Pointer (DpntXX)	9XX	KBから入力されるデータの診断コマンド列内の位置をXXによって示す。
Recieve(RcvXX)	AXX	DIから受けとるデータのバイト数を示す。
Mask(Mask)	803	書き込みデータ、読み出しデータにマスクをかける
Shift Right (ShftXX)	BXX	書き込みデータ、読み出しデータをXXビット右へシフトする
Shift Left (ShftXX)	CXX	書き込みデータ、読み出しデータをXXビット左へシフトする
Output(OutXX)	DXX	データをXX桁 CRTへ表示する
Input(InpXX)	EXX	KBよりXX桁のデータを受けとる
Step	804	EPU を 1サイクル進める
End(End)	80F	手順終了




```
0010        Name
0020        AR
0030        Read
0040        Write Data to MIR(AR → Mbus)
0050        Read Data in Mbus
0060        Rcv04
0070        Shf104
0080        Mask
0090        FF000000
0100        Out02
0110        End
```

図7.1 読み出し手順例

７．３．マイクロ診断プログラムの設計[4]

７．３．１．設計思想

故障検出機構で検出された故障は、修復単位すなわちプリント基板単位に局所化されて交換修理される必要がある。この修復作業は容易でかつ短いことが望ましく、このことはアベイラビリティの向上になる。修復作業の容易さには、マイクロ診断プログラムが正確なこと、取扱いが易しいこと、どんな故障に対しても何らかの修復処理ができること、CPU ハードウェアの変更に柔軟に対処できることが含まれる。

マイクロ診断プログラムの正確さは診断コマンド列で作成されたテストプログラムに依存するので、テストプログラム自身を十分ディバッグすることによって解決できる。マイクロ診断プログラムの実行に際して、装置の繋ぎ換えのような特別な作業は不要とし、かつ SVPの簡単なキーボード入力とニーモニックによる診断結果を画面上へ表示することによって取扱いを容易にする。また、テストプログラムを診断記憶にロードしてPATROLとして実行させたり、単に診断記憶上でループテストとして

実行させたりすることによって間欠故障に対処する。SVPマイクロプロセッサソフトウェアとテストプログラムを分離することによってCPUハードウェアの変更に対してテストプログラム部分のみを変更するだけですむように設計する。

一般にマイクロ診断の診断順序はStart Smallにより実行される[5]。このCPUのA,V,Cサイクル処理ハードウェアは結線論理により制御され、Eサイクル処理ハードウェアはマイクロプログラムによって制御される。また診断部から直接書き込めるレジスタは制御記憶回路周辺レジスタとEサイクル処理ハードウェア内TRレジスタで、診断部から直接読み出せるレジスタ及びバスは制御記憶回路周辺レジスタとMbus及び診断バスである。このことから、SVPをハードコアとして診断部DU、制御記憶周辺回路、Eサイクル処理回路、Aサイクル処理回路並びに制御回路、Vサイクル処理回路並びに制御回路、Cサイクル処理回路並びに制御回路、SCUインタフェイス回路の順序で診断していくのが妥当である[6]。

マルチCPUシステムでは1つのCPUが修復中であり、他のCPUは稼動中であるとき、修復中のCPUのSCUイン

タフェイス回路を診断したいことがある。このとき診断部DU内TMRレジスタのCPU Stand-alone ビットをセットすることによりSCUインタフェイス折り返し機構が動作し、故障CPUを物理的に切り離さないでも診断可能となる。

７．３．２．診断エクゼクティブの設計

CPUハードウェアの変更に対処するために、マイクロ診断プログラムに保守コマンド処理ルーチンと同様に制御語を導入して、マイクロプロセッサアセンブリ言語で書かれた診断エクゼクティブと主に診断コマンド列からなるテストプログラムに分離した。テストプログラムは多数の小さなテスト単位（テストページ）から構成されている。診断エクゼクティブは診断条件の設定、テストページのフロッピーディスクからの読み出し及び実行、診断結果の画面上への表示等の各テストページに共通した処理を行なうように設計する。

保守コマンドDiagの入力によってマイクロ診断エクゼクティブをフロッピーディスクからオーバーレイ領域に読み込んで制御をマイクロ診断エクゼクティブに渡し、

診断エクゼクティブは順次テストページをフロッピーディスクから読み出し診断部DUに診断コマンドを送ることによってマイクロ診断を進めていく。1テストページを1回のフロッピーディスクからの読み込みで可能ならばマイクロ診断プログラム実行時間は次式で近似できる。

$$T \sim t(a)+n(t(b)+t(c)+t(d)) \qquad [7.4]$$

ここで、t(a)はマイクロ診断プログラムの条件設定に必要なSVPのマイクロプロセッサと診断部DUの実行時間の和、nはテストページの数、t(b)は1テストページを実行するのに必要なマイクロプロセッサの実行時間、t(c)は1テストページをフロッピーディスクから読み込む時間、t(d)は1テストページの診断コマンドを実行する時間である。一般に、t(a)<<n(t(b)+t(c)+t(d))であるので、マイクロ診断プログラムの実行時間Tは次式となる。

$$T \sim n(t(b)+t(c)+t(d)) \qquad [7.5]$$

大多数のテストページでは、t(c))t(b)+t(d)であるので、実行時間Tは次式で近似できる。

$$T \sim nt(c) \qquad [7.6]$$

殆んどのテストページにおいて、t(c)はフロッピーデ

ィスクの回転待ち時間とヘッドロード時間の和であり、平均回転待ち時間を80ms及びヘッドロード時間を20msとすると、診断プログラムの実行時間Tの予測値は次式となる。

$$T \sim 0.1n \qquad [7.7]$$

また、任意のテストページを実行するとき、すぐにそのテストページが読み込まれなければならないので、ディレクトリが必要となる。しかし、すべてのテストページのフロッピーディスク上の格納場所を示すディレクトリを持つことは、SVPメモリのオーバーレイ領域が少ないので、容量の点からも、また実行時間の点からも得策ではない。そこで、テストページをテスト番号の小さい順にフロッピーディスクに格納しておき、各トラックの先頭に格納されているテストページのテスト番号をディレクトリとして持つことにする。このときのテストページをオーバーレイ領域へロードする時間t(d)は次式で近似できる。

$$t(d) \sim t(e) + m\ t(f)/2 \qquad [7.8]$$

ここで、t(e)はディレクトリをフロッピーディスクから読み込み該当トラックを捜す時間、 mは 1トラックに

格納されているテストページ数、t(f)は 1テストページをフロッピーディスクから読み込む平均時間である。

1トラックに平均して20テストページ格納するように設計すると、t(e)<<10t(f)であるので、該当テストページをオーバーレイ領域へロードする時間t(d)は次式となる。

$$t(d) \sim 10t(f) \qquad [7.9]$$

フロッピーディスクからの平均読み込み時間t(f)は、トラック移動を伴わないので次式で近似できる。

$$t(f) \sim t(r)+t(l) \qquad [7.10]$$

ここで、t(r)は平均回転待ち時間及びt(l)はヘッドロード時間であり、それぞれ約80ms及び20msとすると、t(d)は約 1秒となる。

このように設計することによって操作性のよいマイクロ診断プログラムを開発することができる。

7．3．3．テストページの設計

CPUのハードウェア変更への対応並びに診断エクゼクティブ及びテストページの簡潔化を考慮して表7.3で示す制御語を導入した。そこで、テストページは制御語と

表7.3　マイクロ診断用制御語

制御語	コード(16 進)	説　　明
Ident.(No.)	800	テストページ番号
PCB Name(Board)	801	被疑基板名
Logic(logic)	802	被疑回路名
Test Data(Data)	803	テストデータ
Recieve(Rcv)	804	DIからデータを受けとる診断コマンド列
Send(Send)	805	DIからデータ受取を伴わない診断コマンド列
Interrupt(Int)	806	正解値(CPUからの割り込みが有る)
No Intr(Noint)	807	正解値(CPUからの割り込みが無い)
End(End)	80F	1 テストページ終了

```
        ** MICRO DIAGNOSTICS **
0100*EJECT
0110C TEST PAGE NO 001430
0120C REGISTER NAME SPM(0)
0130      IDENT
0140       001430
0150      BOARD
0160       EA,EB
0170      LOGIC
0180       TR→SPM(0)
0190      SEND
0200       Write Data to TMR (All Clear)
0210       Read CS
0220       Write Data to TMR (No Clear)
0230      DATA
0240       00000000
0250       FFFFFFFF
0260      SEND
0270       Write Data to TR (DATA→TR)
0280       Write Data to MIR (TR →ALU →SPM(0))
0290       Step EPU
0300       Write Data to MIR (SPM(0) →Mbus)
0310      RCV
0320       Read Data in Mbus
0330      NOINT
0340       FFFFFFFF
0350       00000000
0360       FFFFFFFF
0370      END
```

図7.2. マイクロ診断テストページ例

診断コマンドで構成される。テストページの一例を図7.2に示す。

図7.2において、0100行から0120行目は診断アセンブラへの指令または注釈である。制御語IDENTに続く00143 0はテスト番号で、テストページを捜すときにも使用される。制御語BOARDの次のEA,EBは被疑プリント基板名を表し、制御語LOGICの後のTR→SPM(0)は被検査論理回路名を示す。ここまでは、診断辞書に相当するもので、次の診断手順によって故障が検出されたならば、マイクロ診断エクゼクティブはこれらの情報を編集して画面上に表示する。制御語SENDに続く0200行から0220行目までの診断コマンド列は、次のEPUの駆動に先がけての初期化するためのものである。制御語DATAの後の00000000とFFFFFFFFは直後の診断コマンドのデータである。この制御語以降でかつ制御語NOINTまでにある診断コマンド列は制御語DATAで与えられたデータ分繰返し実行される。すなわち、0260行目の00000000→TR、0280,0290,0300,0320行の診断コマンドが最初に実行される。次にFFFFFFFF→TR、0280～0320行の診断コマンドが実行される。これらの診断コマンド列はTRレジスタに与えたデータをス

クラッチパッドメモリSPM(0)に転送しMbUSで読み取る動作を行う。また、制御語RCV に続く診断コマンド列は読み込みデータを伴う。制御語NOINT の直後のFFFFFFFFはマスク語で続く正解値と観測値とをマスクして比較する。すなわち、00000000をTRレジスタに与えて検査したときの観測値を00000000とし、またFFFFFFFFをTRレジスタに与えて検査したときの観測値をFFF7FFFFとすると、まず(FFFFFFFF).AND.(00000000) と(FFFFFFFF).AND.(00000000) が比較され、次に(FFFFFFFF).AND.(FFFFFFFF) と(FFFFFFFF).AND.(FFF7FFFF) とが比較される。この場合では当然故障が検出されたことになる。　このようにテストページを設計することによって、同じ検査手順に対して異なったデータを与えることができテストページも簡潔になる。なお低級言語で書かれたテストページは診断アセンブラによってビットパターンに変換され、フロッピーディスクに予め格納されている。

７．４．開発結果及び検討

殆んどの保守コマンドの実行時間が 1秒以内になるように設計すると、 1つの保守コマンドの実行ではフロッピーディスクのアクセス回数が最大 4回にしなければならない。保守コマンド処理ルーチンのなかで処理内容が多様なものはREAD保守コマンドとWRITE 保守コマンドである。READ保守コマンドには、レジスタの内容を読み出すもの、主記憶の内容を読み出すもの、制御記憶の内容を読み出すもの及び診断記憶の内容を読み出すものが必要である。これらを指定するためにサブフィールドを設けて次のようなフォーマットとした。

READ/REG/Reg. name　　：レジスタ

READ/MEM/memory address (from - to)：主記憶

READ/CS/memory address (from - to) ：制御記憶

READ/DM/memory address (from - to) ：診断記憶

WRITE 保守コマンドもサブフィールドが必要であり、このようにサブフィールドが必要な保守コマンドはREAD保守コマンドと同様な次のフォーマットにした。

field 0/field 1/--/field m

保守コマンドがキーボードから入力されたとき、保守

プロセッサモニタは、モニタ内に持っているディレクトリからfield 0 に対応する保守コマンド処理制御ルーチンをオーバーレイ領域に読み込んで制御をこのルーチンに渡す。制御ルーチンは内蔵ディレクトリからfield 1 に対応する処理ルーチンをオーバーレイ空領域に読み込み制御をこのルーチンに渡す。以下同様の処理を行う。

このように保守コマンド及び保守プロセッサを設計していった結果、 mは 2までで十分であり、処理の多様なREAD/REG保守コマンド及びWRITE/REG 保守コマンドではフロッピーディスクのアクセス回数は 4回となり、これらの保守コマンドの実行時間は約 1秒であり、実用上問題なかった。

READ/REG/TR という保守コマンドを入力した場合を例にとり、保守プロセッサの動作を説明する。保守プロセッサモニタはREAD保守コマンド処理制御ルーチンをフロッピーディスクから 900～ BFF 番地に読み込んでこのルーチンに制御を渡す。次に制御ルーチンはREG 処理ルーチンを 800～ 8FF 番地に読み込んでこのルーチンに制御を渡す。処理ルーチンはレジスタのディレクトリをフロッピーディスクから 600～ 7FF 番地に読み込んでTRレジ

スタを読み出す手順が格納されているディスクアドレスを得、この手順を 600～7FF 番地に読み込む。次にこのルーチンは手順に含まれた制御語を解釈して診断コマンドを CPUに送った後でデータをCPU から受取り編集してCRT に表示した後、制御を呼んだルーチンに返す。このようにフロッピーディスクのアクセスは 4回であるが、保守プロセッサが使用するオーバーレイ領域は約1.5K語であった。WRITE/REG 保守コマンドの処理も同様であった。なお、READ/REG及びWRITE/REG 保守コマンドでアクセスできるレジスタはそれぞれ 520個と 160個であり、対応する手順を格納しておくフロッピーディスクの容量はそれぞれ 30Kバイト及び 16Kバイトであった。

マイクロ診断プログラムの動作を次に説明する。保守コマンドdiagの入力によってマイクロ診断エクゼクティブの制御ルーチンがオーバーレイ領域の A00～BFF 番地にフロッピーディスクから読み込まれてこのルーチンに制御が渡される。制御ルーチンは初期設定を行うために初期化ルーチンをオーバーレイの空領域 700～9FF 番地に読み込んで制御をこのルーチンに渡す。初期化ルーチンは人間と会話して診断条件を設定した後で制御を制御

ルーチンに返す。次に、制御ルーチンはテストページ処理ルーチンをオーバーレイ領域の 700～9FF 番地に読み込んで制御をこのルーチンに渡す。テストページ処理ルーチンは 1テストページをオーバーレイ空領域の 600～6FF 番地に読み込んで制御語を解釈してCPU に診断コマンド列を送った後で結果を受取り編集して CRTに表示する。テストページを処理するか、途中で故障が検出されたなら、その旨を表示して制御を制御ルーチンに返す。

このようにマイクロ診断プログラムが使用するオーバーレイ領域は約1,5K語であり、 1テストページを 256バイトとすることによって全体のテストページ数は約2100となった。故意に故障を起させて実験したところ、マイクロ診断プログラムの実行時間は約 8分で、仮定した故障の約90%を検出し、このうち約85%を適中する能力があった。しかし、CPU のA,V,C サイクルハードウェアの大部分が結線論理で動作するために、テストページの作成が困難であった。

なお、図7.3 にチャンネルサービスプロセッサ、保守プロセッサ、割り込みプロセッサ及びマイクロ診断プログラムが動作したときのCRT 画面例を示す。

```
     MEDIA-S#12345-06 MNT 0-12-03 MASTER-TAPE
***** CPU INTERRUPT *****
    * REASON --- FAULT ON FAULT *
    * MULTI CPU ? yes
    * AUTO RECONFIGURATION.  CPU#1 RELEASED *
    * PLEASE FIX CPU#1*
     OUT OF PAPER PR 012
(M)*read/reg/tr
(M)   TR=0000FFFF
    MEDIA-S#12345-06 MNT 0-12-03 MASTER-TAPE
(M)*diag
(D) CPU STAND-ALONE ? yes
(D) SELECT TESTS ? no
(D) START MICRO DIAGNOSTICS.
(D) TEST NO. 00100        OK.
(D) TEST NO. 00110        OK.
(D) TEST NO. 00430
(D) FAULT DETECTED!
(D) PCB NAME. EA,EB
(D) LOGIC NAME  TR→SPM(0)
(D) S/B DATA FFFFFFFF
(D) WAS DATA FFF7FFFF
(M)*bye
```

図7.3.　画面例

７．５．おわりに

本章では、48Kビットのプログラム用メモリとフロッピーディスクを使用した操作性の良い保守プロセッサ及びマイクロ診断プログラムの設計思想と、これらを開発して使用した結果について議論した。

保守プロセッサにおいて、保守コマンドのサブフィールド毎にコマンド処理ルーチンの 1モジュールを対応させ、制御ルーチンがサブフィールドを解釈する都度、18Kバイトのオーバーレイ領域の一部に読み込み実行させてコマンド処理を行った。フロッピーディスクからオーバーレイ領域に、プログラム、ディレクトリ及びレジスタ読み出し等の手順の読み込む回数を 4回以内に設計することによって、保守コマンドを入力したときの応答時間を 1秒以内にできた。また、レジスタ読み出し等の手順に簡単な言語（12種類の制御語）を導入することによって、この手順と SVPソフトウェアを分離でき、かつ、レジスタは TR,SPM 等のニーモニックで指定できるようになった。保守コマンドでアクセスできるレジスタは約520個であり、保守プロセッサの全体の容量は約100Kバイトであった。

マイクロ診断プログラムにおいて、1テストページを256バイトに固定し、テストページの実行中は制御ルーチン及びテストページ処理ルーチンをオーバーレイ領域に常駐させることによって、2100テストページの実行時間を約8分にできた。また、テストページに簡単な言語（9種類の制御語）を導入することによってCPUの検査手順とSVPソフトウェアを分離でき、さらに同一の検査手順で検査データ部分を変える制御語によってテストページをコンパクトにでき、マイクロ診断プログラムの全体容量は約550Kバイトとなった。

このような設計で、実用的な保守プロセッサ及びマイクロ診断プログラムが開発できた。

参考文献

1) j.Reilly,A.Sutton,R.Nasser and R.Griscom:Processor Controller for the IBM 3081,IBM Jour. of R and D,Vol.26,No.1,pp22-29(Jan.1982)

2) DEC:VAX 11/780 CPU Technical Description,EK-KA-780-TD

3) 岩根，飯田，西中：オペレーションプロセッサ，昭和52年度信学全大，1356(1977)

4) 岩根，佐藤，鈴木，高橋：ACOS 77/600 CPUのマイクロ診断方式，信学技法，Vol.78,No.26,pp25-33(1978)

5) 萩原：マイクロプログラミング，産業図書(1977)

6) 岩根，佐藤：故障診断に適したディジタルシステムの一構成法，信学技法，Vol.83,No.55,pp1-10(1983)

第８章　おわりに

1970年以後、マイクロプログラム制御方式は、コンピュータを系統的かつ組織的に設計するためにのみ有効ではなくコンピュータ間の互換性を保持するために、また処理を高速化するために、さらに RASの向上のために使用できることが明らかになり、マイクロコンピュータから大型コンピュータに至るまで多くのコンピュータに使用されており、大型コンピュータになるほどマイクロプログラミングの種々の応用が実用化されている。

本論文は、マイクロプログラム制御コンピュータの処理の高速化と RASの向上に関する研究についてとりあげたものである。マイクロプログラム制御コンピュータの開発にあたり、処理の高速化及び RASの向上を実現するために、着想段階から試作にいたるまでの各段階で解決しなければならない課題をとりあげて、 8章に分けて議論した。

第 1章は序論であって、マイクロプログラム制御方式の動作、マイクロプログラミングの歴史、高速化及び RASの向上を図る上での問題点を概説し、本研究の必要性並びに目的について述べた。

第 2章は、マイクロプログラム制御コンピュータの高速化及び RASの向上における設計上の問題点をとりあげて議論し、次の結論がえられた。高速化では、平均命令実行時間とマイクロ命令実行時間分布との関係及びマイクロ命令実行時間分布とのマシンサイクル時間との関係について議論し、マイクロ命令実行時間のばらつきが大きいほど可変長マシンサイクル時間方式が定性的に有利である。 RAS技術の一つであるマイクロプログラムによる検査方式では、制御記憶の容量の増加及びシステムのリソースの使用が問題であり、これらを削減する必要がある。 RAS技術を支えるハードウェアを個々に用意するとコストが増大するので、できるかぎり共通化する必要がある。また SVPは RAS向上の要であり、部品数を少なくして装置自身の信頼度を向上させる必要がある。

第 3章では、完全可変長マシンサイクル時間方式及び量子化可変長マシンサイクル時間方式の最適化について議論した。完全可変長マシンサイクル時間方式においてn種類のマイクロ命令実行時間から最適な m種類のマシンサイクル時間を決定するとき、動的計画法を適用すると、目的関数の評価回数が(m-2)*n**2/2-m*(m-4)*n/2+m

*(m**2-6*m+5)/6 以下になることが判った。量子化可変長マシンサイクル時間方式では、単純なアルゴリズムで短時間で最適解が求まることが判った。

第 4章では、設計時のある仮定によるマイクロ命令実行時間分布に基づいたマシンサイクル時間の最適化とコンピュータの使用環境に基づいたマシンサイクル時間のチューニングについて議論した。設計時の最適化では、シミュレーションにより求められたマイクロ命令実行時間分布に対する各可変長マシンサイクル時間方式の平均命令実行時間を求めて各方式について検討した結果、マシンサイクル時間の種類をある程度以上多くしても性能は殆んど変わらなく、完全可変長マシンサイクル方式が優れておりことが判り、さらにこの方式で試作したところ固定長マシンサイクル時間方式よりも約 1%の費用増加で平均命令実行時間は0.59倍に短縮できることがことが判った。チューニング時において、理論上完全可変長マシンサイクル時間方式は量子化可変長マシンサイクル時間方式よりチューニング効果が大きい、換言すれば量子化可変長マシンサイクル時間方式の方が環境の変化の影響を受けにくいことが判った。2,3 例でもってシミュ

レーションにより性能改善率を求めると、前者は最大 6％、後者は最大 5％程度であった。

第 5章は RAS向上のためのハードウェア構成について議論した。命令実行のために準備されたマイクロオーダと診断コマンドを組み合せることによってハードウェアを共通化し、制御記憶の特定のマイクロ命令を実行するための手順を格納する診断記憶並びに読み出し専用の診断バスを導入し、小規模低速なマイクロプロセッサを SVPに組込むことによって SVP自身の信頼度を向上させると共に診断部と SVPの速度差を利用してインタフェイスを簡単にした。このように設計することによって、 CPUでRAS 関連ハードウェアの占める割合は約 7％となり、SVP のプログラム用メモリ容量は 48Kビットであり、従来の方式よりも改善できたことが判った。

第 6章では、PATROLによる故障検出技術及び SVPを使用する故障回復技術について議論し、次の結論を得た。PATROLによる性能低下は約0.12％であり、ハードウェア量は従来のマイクロプラグラムによる検査法の約1/5 となり、間欠故障の早期検出に有効であることが判った。SVPの CPU故障回復を含む割り込み処理プログラム量は

約8Kバイトとなり、9000ビットをROM領域に格納することにより実用的であることが判った。

第7章では、故障修復に必要な保守プロセッサ及びマイクロ診断プログラムの設計思想と開発結果について議論し、次の結論を得た。保守プロセッサ及びマイクロ診断プログラム共に、CPUの変更に対応できるように、SVPソフトウェア及び診断コマンド列とマイクロ命令からなる手順の保守が容易になるように、さらにこの手順がコンパクトになりかつ開発が容易になるように、この手順に簡単な言語を導入すると共にフロッピーディスクのアクセス回数を少なくすることによって、小規模低速マイクロプロセッサでも実用的なSVPが作成できることが判った。

本論文において得られた研究成果を総括して述べたが、まだ研究すべき問題も残っている。例えば、自動チューニングの実用面での効果並びにコストの定量化の諸問題、検査並びに保守の容易なCPUの回路構成、及び間欠故障の故障個所の指摘等が残っている。これらは、マイクロプログラム制御コンピュータがますます発展していくなかで、重要な課題になると考えられる。

謝辞

本研究の大部分は筆者が東芝株式会社に在職中に行ない、及び豊田工業大学でまとめたものである。

本論文をまとめるにあたって特別の御指導と御激励を賜った京都大学教授萩原　宏先生に深く感謝し心からお礼申し上げます。また本論文をまとめるにあたって御助言を賜った京都大学教授矢島脩三先生に深く感謝します本研究を進める際、数々の御助言を賜った東芝株式会社佐藤文孝氏に深く感謝します。

おわりに、本研究の推進のために御尽力、御協力いただいた東芝株式会社青梅工場の皆さまに感謝します。

本研究に関する発表

＜論文＞

(1) 岩根，佐藤：大型計算機における保全性設計、信学論 Vol.63-D,No. 3(1980)

(2) 岩根、佐藤、溝口：マイクロ命令の実行時間分布に基づく最適マシンサイクル時間の決定法、信学論 Vol.63-D,No.12(1980)

(3) 岩根、佐藤：ある種の割当て問題における解法について、情報処理学会論文誌 Vol.23,No. 2(1982)

(4) 岩根、佐藤：ある分割問題の動的計画法による高速アルゴリズム、情報処理学会論文誌（掲載予定）

(5) 岩根、佐藤：低速小規模マイクロプロセッサを内蔵したサービスプロセッサ、 情報処理学会論文誌（投稿中）

＜研究会資料＞

(1) 岩根、佐藤、鈴木、高橋：ACOS 77/600 CPU のマイクロ診断方式、信学技報 Vol.78,No.26,R78-4(1978)

(2) 岩根、佐藤、溝口：マイクロプログラム制御計算機のタイミング設計における一考察、信学技報 Vol.79,No2

72,EC79-83(1980)

(3) 岩根、佐藤：故障診断に適したディジタルシステムの一構成法、信学技報Vol.83,No.55,EC83-12(1983)

＜著作＞

当麻　編：マイクロコンピュータ応用ハンドブック

7.4 節、計算機のRAS、昭晃堂(1983)

＜講演及び特許＞

電子通信学会及び情報処理学会等　5 件

U.S. PATENT　、　3 件